All the student submissions were automatically submitted to turnitin.com, so they will have scores like 0%, 10%, 43%, etc. Ignore these. I had intended to try to catch students with overly similar answers, but many students got high scores just by repeating the questions.

Question 0A: Do not grade. But do look at the assessment submissions of all team members "together", to see if there are any patterns. Bring to my attention anything that looks like a real problem. I'm going to go through them separately, but pointers from you can speed up dealing with any bad cases (hopefully none).

Question 0B: Do not grade. Ignore this unless there's something alarming, I plan to go through all of them.

Graded questions: There are three graded questions, part 1, part 2 and part 3.

Part 1 (50 points): Consider the service your team has proposed as your team project. Imagine the US federal government has imposed a new rule that all software services operating in the US provide "special access" for the NSA to monitor all activities of all client services and applications using the service. Your service operates in the US and your small company has no means to fight this new rule; however, a large company named "Orange" is fighting the rule in court. Removing your service from the US market is not an option for the purposes of this assessment. Devise features that you would need to add to your service to support the "special access" rule. Assume your development team will implement these features, and the NSA will not modify your codebase or its dependencies in any way. Include a *kill switch* that can be instantly activated to turn off all these features temporarily, even when their use is in progress, should Orange obtain a temporary suspension of the rule from a federal court. Keep in mind you may need to turn all these features back on again at any time, and back off again at any time, as Orange's case works its way through the courts. Include in your answer a link to your team project github repository and make sure the instructor and the IAs have collaborator access.

Part 1 has two graded subparts, A and B.

part 1A (25 points): First briefly state the gist of what your service does. Then write one or more user stories for the new "special access" functionality to be added to your service. These user stories should reflect what "monitoring" the use of your service could mean in the context of the features provided by your specific service. Maybe the NSA wants to know whether foreign nationals are using your service to infiltrate art schools or to trade photos of kittens (if information about art schools or photo sharing is supported by your service, respectively). Finally, write one or more user stories about the kill switch.

The article at https://www.reuters.com/article/us-usa-security-congress-insight/spy-agency-ducks-questions-about-back-doors-in-tech-products-idUSKBN27D1CS (Links to an external site.) explains that such a feature is not as far-fetched as it sounds.

Write at least two user stories, not a single huge user story, at least one about the NSA monitoring and at least one about the kill switch. User stories that do not address the NSA monitoring or the kill switch will not count towards the two. Your user stories must use this format:

< *label* >: *As a < type of user >, I want < some goal > so that < some reason >*

My conditions of satisfaction are < list of common cases and special cases that must work >

User stories were covered in the September 28 lecture.

Grading:

3 points for the student presenting a reasonable description of what their service does. Look up their revised proposal if you don't know, give 0 for this if they don't include a link to their github repo. This repeats below, in case students skip A but do B (or vice versa). If ok for either A or B, then give credit for both.

12 points for at least one user story about an NSA agent (or multiple agents) "monitoring" use of the service, which should include read access to the list of users, any metadata about those users such as countries or IP addresses they have accessed the service from, and those users' service-specific data. Any of these might or might not involve a query or filter, e.g., accessing or flagging xxx specific user rather than all users, yyy kind of user rather than all users, user data about zzz rather than all user data. NSA might want the data in some specific format, such as JSON. Some services might not have any per-user metadata, that's ok. Either a push model (service sends new data to NSA automatically) or pull model (NSA repeatedly asks service for new data) is fine. Dividing these into multiple user stories or combining in one user story is fine, as long as it's a separate story from the kill switch below. 6 of these points for baseline user story, 6 for meaningful conditions of satisfaction about the queries/filters, if any, the users/data accessed, non-interruption of regular user service, etc.

10 points for at least one user story about a service administrator "kill switch" that turns off and on NSA's monitoring. Turning off should totally remove NSA access, in a push model NSA is no longer sent anything, in a pull model NSA can no longer retrieve anything, either way without interfering with regular users. Turning on restores NSA access (it's ok either way whether the user story supports or doesn't support retroactive access for the data missed while turned off). Dividing these into multiple user stories or combining in one user story is fine, as long as it's a separate story from the monitoring above and both turn off/turn on are covered. 5 of these points for baseline user story, 5 for meaningful conditions of satisfaction that should include the status of NSA access.

Do not give any credit for any user stories about their baseline service that do not address NSA monitoring and/or the kill switch.

part 1B (25 points): First write one or more use cases for the new "special access" functionality to be added to your service. These use cases should reflect what "monitoring" the use of your service could mean in the context of the features provided by your specific service. Then write one or more use cases about the kill switch. Your use cases do not need to match your user stories.

You can assume that the NSA will provide a special API for authenticating itself. The article at https://tutanota.com/blog/posts/encryption-backdoor-fails/ (Links to an external site.) gives an idea of how well you should expect that API to work, but use it anyway.

Write at least two use cases, not a single huge use case, at least one about the NSA monitoring and at least one about the kill switch. Use cases that do not address the NSA monitoring or the kill switch will not count towards the two. Each of your use cases should include at least a title, primary actor, preconditions, postconditions, basic flow, and one or more alternative flows. Make sure to label every step in each flow, including the branch points for alternative flows.

Use cases were covered in the September 30 lecture.

Grading:

3 points for the student presenting a reasonable description of what their service does. Look up their revised proposal if you don't know, give 0 for this if they don't include a link to their github repo. This repeats above, in case students skip A but do B (or vice versa). If ok for either A or B, then give credit for both.

12 points for use case (or multiple use cases) about an NSA agent (or multiple agents) "monitoring" use of the service in the sense above for part 1A. 2 of these points for using the NSA-provided authentication API somehow, either in a precondition/trigger or an early step, to make sure the access is really from the NSA. 2 of these points for at least one precondition that sounds plausible such as a previously flagged user doing something with the service (a trigger instead of or in addition to a precondition is fine), 2 points for at least one postcondition that seems plausible, 3 points for a basic flow with multiple steps that look reasonable such as running queries, aggregating queries, converting data among formats, etc., 3 points for at least one alternative flow with a plausible branch point from the basic flow such as when the NSA authentication fails, no

user or data matching an NSA query/filter can be found, or there's some glitch like database or some dependency "down", "times out", etc.

10 points for use case (or multiple use cases) about a service administrator "kill switch" that turns off and on NSA's monitoring in the sense above for part 1B. 2 points of this for at least one precondition concerned with Orange's court case (a trigger instead of or in addition to a precondition is fine), 2 points for at least one postcondition such as status of NSA access or non-interruption of regular user access, 3 points for a basic flow, 3 points for at least one alternative flow with a branch point from the basic flow – turning off might be basic flow and turning on might be alternative flow, or vice versa, or these might be separate stories, alternative flows might also involve some glitch like database or some dependency "down", "times out", etc.

Do not give any credit for any use cases about their baseline service that do not address NSA monitoring and/or the kill switch.

Part 2 (20 points): Recall our in-class exercises discussing the APIs needed to implement a CONTAIN app (**CON**tact **T**racing for instruction **A**ss**I**sta**N**ts). If you do not attend class regularly, watch the videos and read the lecture slides; CONTAIN has been discussed during at least five different class sessions.

Write a set of CRC cards for the functionality of the CONTAIN app as discussed in class (put the content of your "cards" in the document to submit on courseworks, do not use physical cards). All of the CONTAIN app functionalities discussed in class should appear as responsibilities of some component. All the APIs to external services discussed in class as being useful for CONTAIN should appear as collaborators of one or more components. Make sure you write a *set* of CRC cards corresponding to multiple components of the CONTAIN app, not a single huge card.

I collected all the slides about CONTAIN into a separate "Collected CONTAIN slides" file in the shared google drive, inside the lecture notes folder. The common part is:

"CONTAIN, **CON**tact **T**racing for instruction **A**ss**I**sta**N**ts, does "contact tracing" based on IA (instruction assistant) connections. An IA for xxx course is a contact of all the other IAs for xxx as well as for every student who takes xxx. Since an IA is also a student, they are a contact for all other students in every course they take as well as for all the IAs of those courses." The different slides describe

each of the APIs CONTAIN could use from CANVAS, SSOL, and a (hypothetical) attendance-recording drone.

CRC cards were covered in the October 5 lecture.

Grading:

2 points for correct format: each class (noun) has responsibilities (verbs) and collaborators (nouns corresponding to other classes, to APIs, or to some shared resource like “database”).

3 points for using (somewhere) the CANVAS, SSOL and drone APIs as collaborators. The drone might be treated as part of the app rather than a separate API, that’s ok, but it still should be a collaborator of some other class. For example, facial recognition and/or QR codes were intended to be hidden behind the drone API, not part of CONTAIN itself, but its ok if the CRC cards include something about these.

5 points for covering the baseline functionality of using CANVAS to get list of IAs and students in a class and then including IAs as well as students in contact tracing for that class. Could alternatively use SSOL to get list of students in class but only CANVAS knows the IAs.

5 points for using CANVAS or SSOL to trace through all the different classes taken by a student including IAs. This would be easier to do with SSOL per-student data but could be done by aggregating students-per-class data from CANVAS.

5 points for filtering to only consider students and IAs who actually attended the same class session as recorded by the drone. It’s not necessary to cover possible contact between students leaving and students entering a classroom for successive classes in same room.

The CRC cards do not need to address authentication of the CONTAIN user, student privacy, malicious attacks, or any other security/privacy issues. The CRC cards also do not need to consider glitches like SSOL is offline, the network is down, and so on.

Part 3 (30 points, 5 points per principle): Consider **your own implementation** of the Connect-4 game mini-project from the viewpoint of design principles. Use the latest version you submitted; do not change your code. Discuss each of the following six design principles. If your implementation – including the skeleton code - violates the principle, explain how and in what way it does so. If it does not violate the principle, explain how it might be modified to intentionally violate the principle. Your suggested

modifications need to be realistic for the Connect-4 game and its implementation based on the skeleton code, do not add arbitrary features. Include a link to your mini-project repository and make sure the instructor and the IAs have collaborator access.

1. Single Responsibility Principle
2. Open-Closed Principle
3. Liskov Substitution Principle
4. Interface Segregation Principle
5. Dependency Inversion Principle
6. Don't Repeat Yourself Principle

The relevant design principles were covered in the October 7 lecture.

Grading:

Max 5 points per principle, if one of the principles is missing give 0 for that principle. Copying the question with no answer should also be given 0.

What we are looking for is does the student seem to understand the principle (2 points) and can they apply it positively or negatively (3 points). Although the question asks them to apply it negatively, that is, do the bad design rather than avoid doing the bad design, it looks like some students may have instead described how they applied the principle positively, i.e., avoided doing the bad design. That's ok as long as they demonstrate through an example that they understand the principle.

The skeleton itself has many examples of bad design, particularly involving Gameboard.py. It does not use inheritance or interfaces, and there was no reason to add these, so students should be creative about what they could have done to intentionally violate the relevant principles. Any answer that just says something like the Connect-4 implementation does not use inheritance or interfaces so xxx principle does not apply, and that's all, should get 0.

You should not actually need to check their github repository unless they refer to something without enough context or explanation so it does not make sense unless you actually look at the code.