How Could Serious Games Support Secure Programming? Designing a Study Replication and Intervention

Manuel Maarek, Léon McGregor
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh, United Kingdom
M.Maarek,lm356@hw.ac.uk

Sandy Louchart, Ross McMenemy
School of Simulation and Visualisation
Glasgow School of Art
Glasgow, United Kingdom
S.Louchart@gsa.ac.uk
R.McMenemy1@student.gsa.ac.uk

***Abstract*—While developing and deploying software continue to be more broadly accessible, so is the problem caused by these systems' security not being considered enough by their developers and maintainers. We propose to address this developer-centred security issue with serious games (games for which entertainment is not the main purpose) as a means to motivate developers to consider security threats when developing. We have developed a serious game around secure and non-secure programming exercises to investigate if serious gamification helps to improve attitudes or ability with secure programming. We detail the design choices of the game and how it relates to the programming tasks. In particular we present the design choices we made with the intention to replicate a prior study and discuss the tension that arose between replication and intervention. We discuss the results of a pilot study we conducted and present the steps we plan to take going forward into larger studies.**

***Index Terms*—serious games, secure programming, developer centred security, study replication, serious game intervention**

I. Introduction

In recent times software systems are taking an increasing and prevalent role in our everyday lives. These systems are interconnected, ubiquitous and are operating everywhere from the cloud to our mobile devices. They handle our personal and sensitive data such as our credentials or daily records of our activities. The development of software systems is increasingly accessible to anyone, with the availability of software-as-a-service development platforms and the dissemination power of application stores. As a result, the implementation of security functions in the software systems we use might not be done or vetted by security experts or security-aware developers. As an illustration of the issue, a recent study [1] has revealed the extent of insecure code embedded in mobile applications as a result of the misuse of security Application Programming Interfaces (APIs).

This work was part a project funded by the Research Institute in Science of Cyber Security (RISCS) in association with the National Cyber Security Centre (NCSC) and the Engineering and Physical Sciences Research Council (EPSRC).

A. Background and Motivations

Our goal is to address the issue of poor security of widely available software systems by looking at ways to help the "masses" [2] of developers of these systems to become more aware and proficient in code security.

One direction to tackle this issue is to help the developers to protect themselves against introducing such security bugs by improving API usability and making APIs more secure by default (see examples of such approach in Section IV-A).

Another direction, which is the one we are taking, is to motivate developers to build more secure software by engaging them in taking security into account during development. In our work, we are particularly interested in the role that serious games, which are games for which entertainment is not the main purpose, can play to raise security awareness and understanding. We believe that well-designed games can be effective in impacting behaviour. The gamification approach, which consists of turning a domain activity into a game, has been used in the software engineering context (see examples in Section IV-C), and in the cyber security context (see examples in Section IV-D). The aim of our research is to improve, through games, the security of code that developers produce, and to do so with serious games which we believe offer more potential than basic programming-based gamifications.

This paper presents the idea of a study to investigate how serious games could impact developer-centred security, discusses the design choices we made, and analyses the results of the pilot study we conducted. This pilot is the first iteration of our experiment design, and we detail the changes we plan to make consequently.

B. Research Questions

Our research aims to identify if game interventions are effective at enhancing programmers' abilities and understandings of secure programming. We therefore derived two research questions on the comparative impact of serious games on the security aspect of programming.

RQ1 *Does embedding secure programming exercises in a serious game improve the effectiveness of such*

exercises? — When a player of the game completes a secure programming exercise, do they perform better at it than if they had not played the game?

RQ2 *How differently does a serious game impact security-oriented and other programming exercises?* — If a player completed a programming task and a secure programming task, does the benefit from having played the game affect them differently?

In this paper we only discuss the design of an experiment to answer these questions, and therefore do not answer these questions, see Section I-D.

C. Study Replication vs. Intervention

As a starting point to address these research questions, we wanted to ground our work on an earlier study [3]. This base study was primarily aimed at investigating the ability of developers to program securely and at investigating the security impact of the information sources developers use. We chose to structure our experiment on this base study because of its large sample size of participants recruited online. Using a similar experimental setup would allow us to compare and extrapolate our results with the results of the base study. Replicating this prior study would confirm the original results, and verify the effectiveness of our intervention. As part of our goal to relate our findings with this base study we wanted to replicate the following.

- Use the same programming tasks set in the base study.
- Record participants browsing to identify the information sources used, as was done by self-reporting in the base study.
- Ground the study online on the GitHub interface, while the base study had recruited participants on GitHub[1].

While repeating the same programming tasks, we extended the experiment with participants taking up the tasks within a serious game with the intention to observe how the game impacted on their ability to program securely (RQ1). We also extended the programming tasks with non-security focused programming tasks to compare how the game differently impacts security and non-security tasks (RQ2).

One major issue that arose in the design, preparation and pilot execution of our experiment was the tension between the idea of replicating the base study and our introduction of a serious game intervention. Our goal to replicate the base study and our goal to design a game intervention had conflicting requirements which we discuss in the paper as well as the trade-offs with the middle ground solution we implemented.

D. Contributions, Limitations and Plan of the Paper

As this paper presents the first steps in our investigation of the impact of a serious game on developer centred security, it reports the outcome of the first stage of our iterative design. The contributions are as follows.

- Presentation of the design of a serious game for secure programming.
- Discussion of the choices in replicating a prior study while designing an intervention.
- Analysis of the results of our pilot experiment and details the updates in the design of our study.

The work presented in this paper has also some limitations.

- This paper only reports on a pilot study and therefore does not answer the research questions.
- The study in this paper is based on a single game. The outcome of the pilot and further experiment is therefore limited to the impact this particular game may have.
- This paper does not address the game design process which is the focus of another paper [4].

In Section II, we discuss the design of the game and the programming tasks of our study. The design considerations with respect to the replication and intervention are discussed in Section II-D. Then, the design of the pilot study is presented in Section III as well as its results. In this section, we discuss the issues that occurred, including those that arose from the conflicts between replication and intervention. Finally, related works are discussed in Section IV before we conclude and draw future work perspectives in Section V.

II. Design of the Game

The serious game we have designed follows the genre of tower defence video games. The players switch between the game and completing programming tasks. By performing these tasks, they can unlock new content (named upgrades) in the game. The tasks players complete are python programming exercises described in Section II-A. Each of the elements that makes up the game will have some link back to the programming tasks. For example, offering a story element in the game linking back to a security topic covered in a task. Details of these links are given in Section II-B. The game and program development platform are web-based as described in Section II-C. We end the description of the design of the game by discussing the drives for replication and intervention in Section II-D.

The design of our serious game was done collaboratively between game experts and software security experts. This co-design is the topic of another paper [4].

A. Programming Tasks

The six tasks that users can complete to unlock additional upgrades in the game are described in Table I. The three first tasks denoted by [3] are taken directly from the base study and are security-related, while the remainder three are non-security-related and were written by us. The programming tasks are completed using `python3` and are primarily focused on the use of libraries. An experiment participant (playing the game or simply completing programming tasks) can choose whichever order they wish to complete them in, and is not required to complete them all.

The general premise of a task is that a player will complete a python method (given an empty stub method and specification), and then submit it. For example, the *Credential Storage* task gets the player to write a method which takes

[1] https://github.com/

TABLE I
TASK DESCRIPTIONS

Task Name	Motivation & Description
URL Shortener [3]	Design an algorithm that can securely (non-reversibly) generate shortened URLs.
Credential Storage [3]	Securely (salt, hash with good algorithm) store passwords in a database.
String Encryption [3]	Choose a strong encryption algorithm and mode to encrypt a given string.
Image Analysis	Use a library to analyse an image and return its average colour.
Time Tool	Manipulate the *datetime* python library to find the difference between two given times.
Search Replace	Write a python script which can perform some search and replace operations on text.

in a *username* and *password*, and then store it in a database file. This requires them to make use of the `sqllite3` python library. Players can use `print()` calls and view the standard output resulting from the execution, as well as download any changed files (such as a database file) to assist them with debugging their solution. Some tasks also provide output to assist with debugging — such as printing the contents of a database. Section II-C gives more detailed on the programming environment which is illustrated in Figure 2.

B. Tower Defence Game

Our game is designed as a tower defence strategy game. We intend for the player to repeatedly play the same level to earn in-game money which can be used to purchase upgrades that improve the defence. Some upgrades are locked until the programming tasks (from Section II-A) are completed. This is an incentive for players to complete the tasks. As is often the case in such games, the level is difficult to begin with, but should get easier the more upgrades are unlocked. This difficulty is chosen to incentivise unlocking the upgrades which in turn gives players some practice in programming.

Figure 1 shows the tower defence level in action, you can see two of the main gameplay mechanics in action: *creeps* (a) coming in from the top left corners heading towards the *bank target* (b), and fixed position *towers* (c)–(f) which fire upon the creeps or provide other communication facilities depending on their type. The primary interface for playing the game is via a text entry system, to mimic command line interactions.

We use the design of an enemy attempting to rob a bank as a metaphor that abstracts from programming and cyber attackers to a more real-world environment. This provides an alternative perspective to that given by the programming context of the task. The bridge between two different attack contexts exists to increase player motivation.

The main gameplay mechanics (*creeps* and *towers*) and their links to the tasks are as follows.

1) Creeps: The main antagonists of the game, presented as one of either *Basic*, *Hacker*, *Tank* or *Interceptor*. The primary goal of the game is for the player to use the tools at their disposal to prevent the creeps from getting through the level to their target (the bank).

Fig. 1. Screenshot of the game
The screenshot shows the main gameplay level with the gameplay mechanics: (a) creeps, (b) target asset, and (c)–(f) towers, detailed in Table II.

2) Towers: These are fixed bases that will shoot at incoming enemies. We use several integrations to tie the game mechanics and the programming tasks together. These integrations exist in how the towers operate, of which there are 4 types: *Standard*, *Communication*, *Laser* and *Missile*. Details for each tower are given in Table II.

3) Upgrades: As the player plays the main level they unlock in-game currency. This can be used on the main menu to purchase upgrades which enhance in-game abilities. Some of these upgrades can only be unlocked by completing programming tasks. This is done in order to motivate players to complete the tasks in the order that will best benefit their play style. In order to equally incentivise completing tasks and unlocking upgrades, each of the upgrades will have a different impact on gameplay, making it easier to progress further in a level. These tasks are numbered in the game and both named and numbered in the web browser interface. This is done so that it is easier to see which task corresponds to which upgrade. The numberings are the same for all participants. The various upgrades, as well as the link they have to a given task, are given in Table III.

C. Game and Coding Platform

The game and the code editor are both hosted online and served through a web browser with a special browser extension installed.

The code editor is based on the GitHub website and interface, with additional functionalities provided by our web

TABLE II
TOWER DESCRIPTIONS

Tower	Concept	Motivation & Description
Standard (c)	Credential Storage (Passwords)	This tower is the basic tower that simply fires continually at any creeps that pass in range. However, hacker creeps can disable these towers by "guessing" the password. The player then needs to manually reset the password for each tower that this occurs to. While not explicitly mentioned, longer passwords take longer to hack through.
Communication (d)	String Encryption	These towers act as support for other towers. The player can draw a line of communication between one of these towers and another to increase its range. The link to encryption is that a creep could intercept communications if they are not properly encrypted, represented by the connecting line being broken.
Laser (e)	Credential Storage (SQL Injection)	This tower is a "fall-back defence" — one that is used as a last resort. It can fire a high damage laser at a specific creep, though it has a cooldown timer during which it cannot be used. To target a creep, the player must type its name. However, some creeps have names which could affect a system vulnerable to SQL injection. Typing such names would have undesired effects such as disabling towers.
Missile (f)	URL Shortener	This tower also acts as a "fall-back defence". In this tower's case, to target a creep you need to select an area on the map which will generate a code referencing this location, which can then be typed as a target. These codes can be long however and can be shortened with an appropriate upgrade. Some creeps can guess (through reversibility) these codes, so that if a player enters such a guessed code, the missile will be fired at that creep instead of the target location. The primary method for selection is done using a command-like interface.

TABLE III
UPGRADE DESCRIPTIONS

Upgrade	Task	Motivation & Description
Tower password strength	Credential Storage	Stronger and safer credential storage makes it more difficult for an attacker to guess passwords. This upgrade increases tower resistance to hacker creeps.
Input sanitisation	Credential Storage	Databases must be resistant against injection attacks (such as via SQL). This upgrade prevents creeps with injectable names from interfering when their names are typed for targeting using towers.
Range upgrade	Image analyser	By using image analysis, the standard and communication towers can see better at further distances. This allows them to increase their range.
Improved sensors	Image analyser	With better image analysis, towers can spot any creeps that are trying to obscure themselves using camouflage.
Encryption strength	String Encryption	With stronger encryption, it takes a longer time for creeps to be able to intercept and break communication lines between towers.
Search algorithms	Search & Replace	Once this upgrade is enabled, towers can use different targeting algorithms. This is made possible by the developed algorithm being able to search through the list of creeps in a towers range.
Tower fire rate	Time Tool	Once developed, the time tool allows towers to increase their rate of fire by being able to calculate timing events with more precision.
Missile code length	URL shortener	For the missile tower, a user needs to enter a long code to indicate the location to fire at. Unlocking this upgrade by developing the URL shortening algorithm allows these codes to be shortened, making the tower easier to use.
Missile code reversibility	URL shortener	One issue with codes is that of reversibility: that creeps could guess the location codes, name themselves this, and divert attacks away from intended locations. This upgrade prevents this, as reversibility is a security concern in the programming task.

browser extension. A screenshot of the kind of view the players would see while performing a programming task is given in Figure 2. When editing the code (1) for the programming tasks, the players can also run the code (2) and see the resulting standard output (3) and for certain tasks, any changed files (4). Once the player marks a version of code as the final version (5), they receive a short code that can be used to unlock upgrades in the game, as detailed in Section II-B3. We intend for the player to go back and forth between programming and playing the game. This is done as the game is designed with certain prompts for programming securely, e.g., SQL injection being a game play component (see the *Laser* tower in Table II).

The game itself was created using the *Unity* game engine and exported to HTML5, as part of our aim for it to be playable without needing to install any special extra components. The game was designed to have low system requirements to allow lower specification devices to run it. The game will capture some analytics about how players are playing the game.

By asking participants to install a browser extension, we can:

- Collect information about the domains of websites they visit during completion of programming tasks;
- Modify the GitHub code editor to allow running python code remotely;
- Return a code that could be used to unlock content in the game upon completion of a task.

The code editor itself is the standard editor made available on the GitHub website, with a few extra options for running code and marking a task as complete. This can be seen in Figure 2.

We handle the execution of the python code on our own server for multiple reasons:

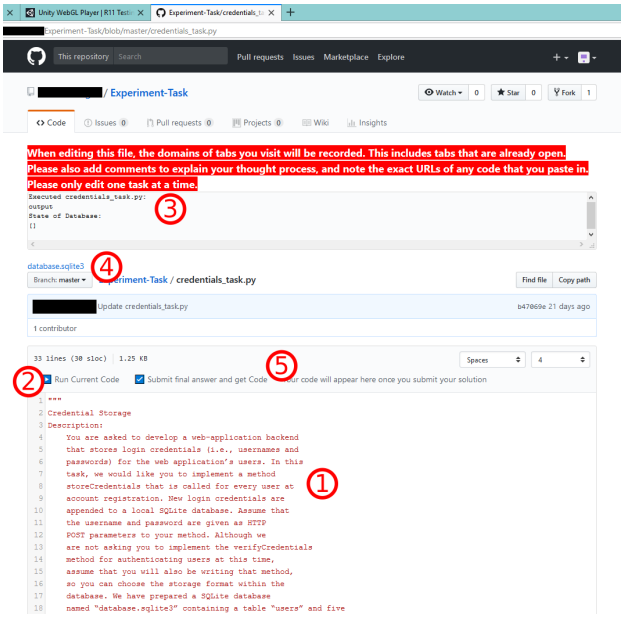

Fig. 2. Screenshot of the program editor

The screenshot shows the program editor of the game which extends GitHub's website and interface with (1) programming tasks, (2) a run code button, (3)–(4) standard output and files resulting from the run, and (5) a final submission button.

- Removing the requirement for the player to have `python3` installed on their system;
- Allowing us to capture the different versions of code that are run;
- Enabling behind the scenes testing to see if the code meets functional requirements.

D. Replication vs. Intervention

In order to replicate the base study [3], we needed to build a platform that could operate in a similar way to the one the base study used [5]. One of the first conflicts between our replication and intervention was that we did not re-use the same platform named *Developer observatory* [5], instead opting for a browser extension that could modify the GitHub web interface and a web-based game. This was primarily done to allow web access to our platform.

The base study asked participants to make note of which sites they visited or obtained code from, and to mark these as comments in their code. Our implementation goes further in collecting the actual domains visited.

In the base study [3] the only information made available to the participants was the specification they needed to implement and the python libraries they could use. One issue in designing our intervention to be compatible with the replication was to keep this level of information given to the participants. This goes against the usual design practice for a serious game, where the objective is to teach something and therefore to provide guidance or adaptive information depending on the player's abilities.

Behind the scenes testing was done on the programming solutions submitted to the six tasks by the player. The result of the tests can be graded automatically and result in different quality of game upgrades depending on the solution quality. This would act as incentive to perform well and be at the art of an educational game intervention. However, with the intention to replicate the base study, our initial choice was to disable making distinction between a correct and incorrect solution to keep the amount of feedback the participants receive similar to the base study.

III. Pilot Experiment

In this section we detail the plan, execution and analysis of our pilot experiment with undergraduate students playing the game and taking up the programming tasks. In our analysis of results, we discuss some of the issues that arose, notably in the design of the tasks and game, and the conflicts between replication and our developed intervention.

We designed the experimental platform so that participants are either invited to play the game and take up the programming tasks within the game, or are asked to complete the programming tasks without the game (control group). This control is to identify if the game itself is effective (RQ1), and to verify if there are any biases of difficulty in regular programming when assessing RQ2. The distribution of participants between the game group and the control group is done with a *round robin* mechanism. For our pilot experiment, we removed the control group to focus on identifying issues with the workflow and interactions between the game and the programming environment.

A. Pre-Pilot Testing

We performed pre-pilot testing, and found no major technical issues. This testing indicated issues with the layout of the questionnaire, which we fixed before starting the pilot.

B. Method

For this pilot, our participants were undergraduate students within the computer science department at Heriot-Watt University. We recruited eight students who had prior experience or knowledge of python programming. The participants were invited to go through the same sign up process as would be used in a full experiment. Once accepting to take part, this involved: installing the browser extension, playing the game, completing some coding tasks and finishing by completing a questionnaire. The participants were encouraged to play as much as possible with the game, and not necessarily following any particular task order. The participants could stay for as long as they wanted. Prior to taking part and in accordance with the ethical procedures in our University, the participants were provided with information on the research, what participating entailed and what data would be collected. They were asked to complete a consent form. Participants could withdraw at any time.

C. Data Collection

In the experiment, we aim to collect data from multiple sources: questioning, game and task analytics, and end-of-session questionnaire. As we conducted our pilot study in person, we could also observe the participants as they were completing the task, and answer any questions they had. We noted issues or positive outcomes observed.

While participants are playing the game, game analytics are recorded on how they play and progress. As they complete the programming tasks and ran code, the back-end system collects the changes they are making to their code.

The end-of-session questionnaire is composed of questions about each task and its integration in the game, about the level of experience in python programming and in security, about gaming experience and perception of the game, as well as some general demographics. This questionnaire replicates the one used by the base study [3] with additional game-related questions.

In the full study, we plan to use the responses from the questionnaire, and the collected programming artefacts (source code, code executions), to determine if the game has had a positive impact on secure programming (RQ1). We also plan to use this data to see if secure and non-secure programming tasks are affected differently (RQ2). This will require comparison of results both between secure and non-security related tasks and also between groups of participants who did and did not play the game.

D. Pilot Results

This section analyses the results from the questionnaire and the observations made during the pilot experiment. Of the 8 participants, only 1 attempted every task (although did not fully complete any of them). The rest of the participants each attempted at least 1 task and completed the exit survey. However, of the 8 that took part only 3 participants ended up with a solution that they submitted as "complete". The base study [3] noted that using python APIs can be hard — evidenced by fewer correct solutions and a high assumed perceived difficulty level. Conducting a small in person pilot study may account for the 0 dropout rate. Our experiment had more tasks than the base study, so it is not surprising in hindsight that the participants completed fewer tasks. Also, worth noting is that unlike the base study, our pilot participants were all students, not active developers, which would likely make the tasks harder. In the base study participants spent "median 56 minutes" on programming tasks [5]. In our pilot, there was an average of 44 minutes spent playing the game and completing tasks (not including the time spent answering the questionnaire). The low task completion rate and the lower time spent on our pilot may be due to the fact that the participants were students with commitments and that the pilot was done in person rather than at their leisure.

1) Questionnaire Results: The questionnaire is composed of questions which related to specific tasks, and some which referred to the game side of the experiment on its own.

The questionnaire answers indicate that, although players feel the game does prompt them for correct solutions, it does not prompt them well to have a secure solution. We also see that in its current iteration, players feel the game is not very useful at helping them to complete tasks. The game's story is also difficult for players to maintain an interest in. There is no strong consensus as to whether the link between tasks and game is strong enough. From the questionnaire's open answer section, responses indicate a lack of time to complete the tasks and play the game.

2) Observation Results: From observation, it was noted that despite the inclusion of a replayable tutorial level which covered how to play the game, some participants still required additional explanations for certain concepts, such as how communication towers were supposed to connect. It was also observed that some players wanted to get through the tutorial as quickly as possible. This indicates an interesting problem of how to offer sufficient explanation while also not trying the patience of players.

E. Pilot Discussion

1) Technical Setup: Analysing the players' ability to simply take part in the experiment shows that it went well (after they were able to sign in following some setup issues). Our plan was for the set up to be as simple as possible. There were several steps between registering or installing the extensions before being able to start taking part. Each step of friction could reduce the number of people taking part. Despite efforts such as using a web-based game to avoid downloads, the implementation was convoluted. This was largely due to two factors: the browser extension which required extra setup steps; and the integration within GitHub which forced us to use an extension in order to modify the interface to embed code executions and outputs. We mention in Section III-F1 how we plan to address these issues.

All the participants were able to start and play the game, even on lower-end machines that were provided during the pilot study. Once correctly signed in, all the participants were able to interact with the webpage offering a code editor. They could write code, run it and see output, and then select a final version as their solution.

2) Tasks: The biggest issue we encountered lay with the programming tasks that the participants had to complete — in terms of difficulty and the time commitment required. Some of the programming tasks can be quite demanding, and this could be a big factor in preventing players from progressing through the game, or put them off completing tasks. At the current level of difficulty, even if developers are capable of completing the tasks, it would still take a very long time to complete them all while still leaving time to play the game. This pilot study gave only a relatively small window for participants to take part, with them spending an average of 44 minutes on the tasks and game with some time for the questionnaire. Modifications to our experimental setup, such as targeting a longitudinal study are detailed in Section III-F2

3) Game Design: Somewhat related to the issues with the programming tasks, we noted some flaws in the design of the game itself. Such as the link between the game, the tasks and the help that the game gives from a programming perspective. The tasks had an influence on the design of the game, however not all respondents felt the presence of the intended link between the tasks and game. There were broad and mixed responses from all the participants as to whether the game had a good link, and whether it engaged them well. This could be due to the fact that players are taken out of the game to complete the tasks, and they may see this as two separate experiences. The only link back to the game at that point is the code the player receives to unlock upgrades. Another issue in the game is that it should have given more help/training for the programming tasks. As mentioned in Section II-D, in a serious game, guidance should typically be provided to meet the learning objectives. This was not done because part of the experiment was initially intended to mirror the base study, and giving too much help in the game could have voided such a comparison. As a result, the game was not offering enough guidance from a perspective of reinforcing secure programming or correct API usage. The in-game tutorial also proved to be moderately useful in explaining how game features worked, though some participants were observed to have still missed the explanations given, requiring additional help to understand how the game was to be played. Changes to the game design will be detailed in the following Section III-F3.

F. Updated Experiment

This section details out our plan for an updated experiment design, taking into consideration the issues discovered during the pilot.

1) Technical Setup: To simplify the login process we are now developing an alternative version which would rely on a GitLab[2] server we will host rather than the GitHub website. This makes the process of installing a web browser extension and linking to GitHub unnecessary. This would however prevent us from recording participants' browsing history while taking up the programming tasks, so instead we will introduce a more explicit mechanism for participants to self-report visited URLs, such as a text box on a page where players can submit visited URLs. With a GitLab server, we can modify the interface as we wish without the need for extensions, and any logins would happen through more traditional means. This will reduce friction to 1) lower the chance of participants dropping out, and to 2) give participants more time to focus on the game and programming.

2) Tasks & Experimental Setup: As mentioned, the programming tasks are difficult, especially with regards to participant time. One solution to this would be to remove some of the tasks from the intervention, however this would reduce the type of tasks and security topics we could cover. An alternative solution would be to simplify the tasks, though this would conflict with our ability to closely compare results as part of a study replication of [3]. The solution we choose to implement in our update is to get participants to play the game over a longer period of time, rather than in a single sitting. A leader board could be used to motivate participants to continue playing the game, and offer additional motivation for completing tasks to do better in the game. This method will require a questionnaire to be offered for each task after it is completed. This will allow us to simplify the questions to focus on one task at a time making it easier to understand, and have the experience fresh in participants minds. General demographic questions would be asked up-front and game-related general questions would need to be asked at the end or at regular timed intervals. Another change we plan to make is to modify how tasks are displayed to participants when they are choosing which to complete. Right now, they are numbered 1–6 and do not change. Instead, these numbers will be removed, and the tasks will be randomly ordered when being shown to participants. By doing this randomly we can avoid issues where participants learn the same concepts from doing the same tasks. This will offer a better investigation of how the intervention, rather than the tasks themselves, are helping the participants.

3) Game: In the updated experiment a re-balance of the information that is provided will be considered. This change in the intervention is a necessary trade-off against the replication. It will make completing an exact comparison or replication of earlier study impossible, but it will allow the game to function more as a serious game intervention. With the game now acting as a proper intervention, we will be able to evaluate its effectiveness at helping players to understand secure programming (RQ1) and through our control see if there is a different impact for non-secure programming tasks (RQ2). The in-game help system may also need to be tweaked in order to provide more assistance to players while avoiding the requirement for a lengthy tutorial session before playing.

G. Summary of Pilot

Our pilot experiment was useful, in that it revealed issues which we can work to improve upon. We need to re-think the programming tasks and how they link to the game. We also need to consider the trade-offs between an exact replication of [3] and designing a serious game intervention teaching secure use of python APIs in an experiment based on the earlier study.

IV. Related Work

As mentioned in the motivations, we wish to tackle the issues surrounding program security from the level of the developer. There are many issues that developers can face, including API usability, information sources available to developers, and developers' lack of interest in taking security into account. In this section, we discuss the related works investigating these issues, as well as other games and game interventions in related domains.

[2]https://gitlab.com/

A. API Usability

An important research into helping developers program more securely is looking into usable security. Research has revealed usability issues in frequently used concepts like *SSL* [6], [7] or libraries such as *JSSE* [8] and *bouncycastle* [9]. These usability issues exist in the APIs themselves, in the documentation around them, or in their level of abstraction. All these issues can contribute to developers producing more security bugs when they use these libraries.

Suggestions have been made that API designers should adopt best practices [10] like making APIs secure by default or making APIs deliberately difficult to misuse. Another suggestion [11] is that API developers should consider an application developer as they would an attacker, and harden their codebase against misuses accordingly.

The concept behind usable API security is that API developers need to take more responsibility in making applications secure. The focus of the security tasks presented by [3] and our game is on security related libraries offered by python. Such a game could be modified to focus on one particular library, educate on its secure usage, and assist in navigating and understanding its documentation.

B. Developer Information Sources

As we noted in Section I-C, the primary aim of the study we try to replicate [3] was looking at the sources of information developers use when programming. A series of studies [1], [12]–[14] on this topic have found that many applications are being published with bugs that can be traced back to Q&A community websites such as *Stack Overflow*.

In this context, a game could play the role of an alternative documentation system, or an interactive portal for information resources.

C. Games for Software Engineering

The idea of basic gamification has been used in software engineering for some time, but [15] claims that we should go further with interventions (adding fictional narrative, adding multiple game elements) if we are to best impact the developer players. Our developed game adopts this idea, going beyond simple achievements and actually framing a story with the aim of better engaging the developer players.

Serious games have been created that overlap with software engineering. As proposed in [16], a study looked at developers and tried to map their process onto a game by using achievements, storytelling and Role-Playing Game (RPG) elements. The development team took interest in the gamified development process. This way of using a game to reward programming is similar to our game though we only focused on specific programming tasks rather than the whole development cycle.

Code Hunt [17], [18] is a programming game. It gets players to develop code that matches a hidden implementation — with only some predefined tests to guide them. However, it offers more of a framework around testing code, and can be used in a contest environment, in addition to teaching programming.

Contests, notably involving Capture the flag (CTF) type objectives, are used to give developers an opportunity to act as programmers and hackers in a competitive manner. One such example, the *Build it, Break it, Fix it* contest [19] gets developers to build a solution, attempt to break another team's build, and then repair their own if it was broken. These types of contest have a more involved security aspect than the game we developed, and suggest an interesting potential future extension: our game could let multiple players assess (and try to break) the secure programming solutions rather than leaving it up to an automatic or expert tester. We are exploring the potential of peer-testing as a means of providing such expert programming peer-feedback for computer science education in [20].

Another such game, *Code Defenders* [21]–[23] puts a game around the topic of mutation testing. Players are taught the concepts of mutation testing by either writing test cases to catch new mutants or writing mutations to the code to escape the new tests. Though our game has testing behind the scenes to evaluate solutions, this testing is neither the focus of what is taught, nor is it transparent to the players.

Both *Code Defenders* and *Code Hunt* are web-based and place the player into a code editor, but they do not include a traditional video game experience. In our game, both elements are present although separated. This separation could be the reason why players felt a disconnect between the game and tasks in our pilot.

Gamifications have also been built into Integrated Development Environments (IDEs). One example [24] presents a tool that motivates developers to remove warning messages by gamifying it through awarding or removing points for bugs created or fixed. This type of game focuses more on simply getting developers to avoid warnings, which might not trigger deeper understanding and could be of little help when the developer produces bugs for which static analysis and compilers do not warn about.

D. Games for Cyber security

Several games have been created for helping players to understand cyber security concepts, beyond just the secure programming that the paper focuses on.

The *hACME* game [25] is a web-based game that gets the player to hack the site that is hosting it. This is less programmer-focused and more hacker-focused, as it has different levels each requiring the player to investigate the site (such as inspecting its HTML source) to "unlock" the next level.

A table top game is presented in [26] that involves all members of an organisation and teaches concepts behind social engineering. Simulating a working environment provides a nice real-world grounding for people to play on, and a way to engage players to think about security. Another table top game [27] is designed to be used in classrooms and is based around having discussions of a simulated company under attack. A table-top Lego game [28] which simulates

an industrial security environment, was found by managers of industries to offer a real educational value.

A common theme of these games is that they simulate a realistic environment. A way to increase the engagement of games like the one we presented here, would be to place it in the context of an existing or simulated organisation's codebase.

V. Conclusion

This paper demonstrates and discusses how serious games could be designed to gauge and impact software developers' security motivations. After designing a serious game and linking it with a framework for programming tasks, we ran a pilot study with a view to investigate if the game is effective at prompting players to program more securely. The pilot revealed issues with the design of the game and the tasks, and indeed the experimental setup as a whole. Our investigation aims to research if serious games can assist programmers in completing programming tasks, and if serious games have different effects on security and non-security related programming tasks. Our pilot does not provide an answer to these questions, but we detail the changes we plan to make to our experiment before conducting a larger scale study in the future. We discussed in this paper both the design of the game, its relationship with the secure programming tasks and the impact our attempt to replicate the base study had on the effectiveness of our serious game intervention.

Online Version: The game is available online at the following address for people to play it.

http://www.macs.hw.ac.uk/games-dcs/

Future Work: We plan to re-work the experiment as referenced in Section III-F. Once the changes are accomplished, we plan to run the experiment an see if the game can be deployed in a teaching environment. This would give us more opportunities to observe the interactions within the game and outside among student developers.

An additional direction is to develop a different game with the same experimental setup. This would allow to investigate whether certain types of game design affect engagement of a player differently in secure programming training.

Possible future alterations for the structure of the experiment could be to remove the current tasks and instead look at certain specific topics that could be taught, for example building tasks that all focus on one specific library. The game could also be deployed for use in a real or realistic codebase, rather than having discrete tasks to raise engagement and to adapt to specific contexts.

VI. Acknowledgement

We would like to thank the anonymous reviewers and shepherd for their constructive comments and suggestions.

References

[1] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl, "Stack Overflow Considered Harmful? The Impact of Copy amp;Paste on Android Application Security," in *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017, pp. 121–136.

[2] C. Weir, A. Rashid, and J. Noble, "Reaching the Masses: A New Subdiscipline of App Programmer Education," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 936–939.

[3] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl, "Security Developer Studies with GitHub Users: Exploring a Convenience Sample," in *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, 2017.

[4] M. Maarek, S. Louchart, L. McGregor, and R. McMenemy, "Co-created Design of a Serious Game Investigation into Developer-Centred Security," in *Games and Learning Alliance*, ser. Lecture Notes in Computer Science, M. Gentile, M. Allegra, and H. Söbke, Eds. Springer International Publishing, 2019, pp. 221–231.

[5] C. Stransky, Y. Acar, D. C. Nguyen, D. Wermke, D. Kim, E. M. Redmiles, M. Backes, S. Garfinkel, M. L. Mazurek, and S. Fahl, "Lessons Learned from Using an Online Platform to Conduct Large-Scale, Online Controlled Security Experiments with Software Developers," in *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*. Vancouver, BC: USENIX Association, 2017.

[6] M. Ukrop and V. Matyas, "Why Johnny the Developer Can't Work with Public Key Certificates," in *Topics in Cryptology – CT-RSA 2018*, ser. Lecture Notes in Computer Science, N. P. Smart, Ed. Springer International Publishing, 2018, pp. 45–64.

[7] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 38–49.

[8] C. Wijayarathna and N. A. G. Arachchilage, "Why Johnny can't develop a secure application? A usability analysis of Java Secure Socket Extension API," *Computers & Security*, vol. 80, pp. 54–73, Jan. 2019.

[9] ——, "Why Johnny Can't Store Passwords Securely?: A Usability Evaluation of Bouncycastle Password Hashing," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, ser. EASE'18. New York, NY, USA: ACM, 2018, pp. 205–210.

[10] M. Green and M. Smith, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security Privacy*, vol. 14, no. 5, pp. 40–46, Sep. 2016.

[11] G. Wurster and P. C. van Oorschot, "The Developer is the Enemy," in *Proceedings of the 2008 New Security Paradigms Workshop*, ser. NSPW '08. New York, NY, USA: ACM, 2008, pp. 89–97.

[12] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers Need Support, Too: A Survey of Security Advice for Software Developers," in *2017 IEEE Cybersecurity Development (SecDev)*, Sep. 2017, pp. 22–26.

[13] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "How Internet Resources Might Be Helping You Develop Faster but Less Securely," *IEEE Security Privacy*, vol. 15, no. 2, pp. 50–60, Mar. 2017.

[14] ——, "You Get Where You're Looking for: The Impact of Information Sources on Code Security," in *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016, pp. 289–305.

[15] T. Barik, E. Murphy-Hill, and T. Zimmermann, "A perspective on blending programming environments and games: Beyond points, badges, and leaderboards," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sep. 2016, pp. 134–142.

[16] E. B. Passos, D. B. Medeiros, P. A. S. Neto, and E. W. G. Clua, "Turning Real-World Software Development into a Game," in *2011 Brazilian Symposium on Games and Digital Entertainment*, Nov. 2011, pp. 260–269.

[17] T. Xie, J. Bishop, N. Tillmann, and J. de Halleux, "Gamifying Software Security Education and Training via Secure Coding Duels in Code Hunt," in *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*, ser. HotSoS '15. New York, NY, USA: ACM, 2015, pp. 26:1–26:2.

[18] J. Bishop, R. N. Horspool, T. Xie, N. Tillmann, and J. De Halleux, "Code Hunt: Experience with Coding Contests at Scale," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, May 2015, pp. 398–407.

[19] A. Ruef, M. Hicks, J. Parker, D. Levin, M. L. Mazurek, and P. Mardziel, "Build It, Break It, Fix It: Contesting Secure Development," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and*

Communications Security, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 690–703.

[20] M. Maarek and L. McGregor, "Development of a Web Platform for Code Peer-Testing," in *The 8th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at SPLASH 2017.*, 2017.

[21] B. S. Clegg, J. M. Rojas, and G. Fraser, "Teaching Software Testing Concepts Using a Mutation Testing Game," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, ser. ICSE-SEET '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 33–36.

[22] J. M. Rojas and G. Fraser, "Code Defenders: A Mutation Testing Game," in *2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, Apr. 2016, pp. 162–167.

[23] J. M. Rojas, T. D. White, B. S. Clegg, and G. Fraser, "Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game," in *Proceedings of the 39th International Conference on Software Engineering*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 677–688.

[24] S. Arai, K. Sakamoto, H. Washizaki, and Y. Fukazawa, *A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools*, Dec. 2014.

[25] O. Nerbråten and L. Røstad, "hACMEgame: A Tool for Teaching Software Security," in *2009 International Conference on Availability, Reliability and Security*, Mar. 2009, pp. 811–816.

[26] K. Beckers and S. Pape, "A Serious Game for Eliciting Social Engineering Security Requirements," in *2016 IEEE 24th International Requirements Engineering Conference (RE)*, Sep. 2016, pp. 16–25.

[27] R. Ottis, "Light Weight Tabletop Exercise for Cybersecurity Education," *Journal of Homeland Security and Emergency Management*, vol. 11, no. 4, pp. 579–592, 2014.

[28] S. Frey, A. Rashid, P. Anthonysamy, M. Pinto-Albuquerque, and S. A. Naqvi, "The Good, the Bad and the Ugly: A Study of Security Decisions in a Cyber-Physical Systems Game," *IEEE Transactions on Software Engineering*, 2018.