
# Let's Hash: Helping Developers with Password Security

Lisa Geierhaas and Anna-Marie Ortloff, *University of Bonn;*
Matthew Smith, *University of Bonn, FKIE Fraunhofer;*
Alena Naiakshina, *Ruhr University Bochum*

https://www.usenix.org/conference/soups2022/presentation/geierhaas

This paper is included in the Proceedings of the Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022).

August 8–9, 2022 • Boston, MA, USA

978-1-939133-30-4

Open access to the Proceedings of the Eighteenth Symposium on Usable Privacy and Security is sponsored by USENIX.

# Let's Hash: Helping Developers with Password Security

Lisa Geierhaas
*University of Bonn*

Anna-Marie Ortloff
*University of Bonn*

Matthew Smith
*University of Bonn, FKIE Fraunhofer*

Alena Naiakshina
*Ruhr University Bochum*

## Abstract

Software developers are rarely security experts and often struggle with security-related programming tasks. The resources developers use to work on them, such as StackOverflow or Documentation, have a significant impact on the security of the code they produce. However, work by Acar et al. [4] has shown that these resources are often either easy to use but insecure or secure but hard to use. In a study by Naiakshina et al. [44], it was shown that developers who did not use resources to copy and paste code did not produce any secure solutions at all. This highlights how essential programming resources are for security. Inspired by the Let's Encrypt and Certbot that support admins in configuring TLS, we created a programming aid called Let's Hash to help developers create secure password authentication code easily. We created two versions. The first is a collection of code snippets developers can use, and the second adds a wizard interface on top that guides developers through the decisions which need to be made and creates the complete code for them. To evaluate the security and usability of Let's Hash, we conducted a study with 179 freelance developers, asking them to solve three password programming tasks. Both versions of Let's Hash significantly outperformed the baseline condition in which developers used their regular resources. On average, Let's Hash users were between 5 and 32 times as likely to create secure code than those in the control condition.


*USENIX Symposium on Usable Privacy and Security (SOUPS)* 2022.
August 7–9, 2022, Boston, MA, United States.

## 1 Introduction

It is well known that end-users struggle with password security. Recent work in the field of Usable Security for Developers and many real-world compromises have shown that many developers also struggle when tasked with implementing password-based authentication systems [7, 18, 29, 40–44, 49]. Unlike end-users' problems that can be dangerous enough, although only one account is usually affected, millions of accounts can be affected if developers make only one mistake.

There have been multiple studies to advance the understanding of how the usability of APIs affects security during software development [4, 22, 33, 40, 42, 43, 45, 68]. However, one crucial aspect is the quality of the available documentation that developers use to solve their tasks. These are often either easy to use but insecure or secure but hard to use [5, 6, 27, 68] with many examples showing that developers copy and paste insecure code from online resources [4, 5, 24, 27]. Acar et al. write [4]: *"our results confirm that API documentation is secure but hard to use, while informal documentation such as Stack Overflow is more accessible but often leads to insecurity."*

So copy and pasting of insecure code is a serious concern to software security, with Fischer et al. [27] postulating that Stack Overflow is harmful. However, studies by Naiakshina et al. [43, 44] show that only the participants who used copy and paste achieved any security. Those who did not use copy and paste did not achieve any security. So while copy and paste has been reliably identified as a serious security threat, it is also an essential method for secure solutions. Thus the goal needs to be to create programming resources that are easy to use but also help developers create secure solutions.

In this paper, we create Let's Hash, a programming resource to aid developers in creating secure code for password-based authentication. Our goal is to offer something as easy to use as Stack Overflow but as secure as official documentation or programming books. We created two versions of Let's Hash. The first is a simple website offering code snippets in a similar style to Stack Overflow. With this version, developers

are still required to select and assemble the code snippets themselves. The second adds a wizard on top, which lets developers specify the security requirements, and the wizard assembles all the necessary code, which is then ready to use.

Currently, Let's Hash can help developers create code for the following three tasks: hashing and salting passwords for storage, creating and enforcing password policies, and complementing password-based authentication with a second factor; two-factor authentication (2FA). To evaluate the usability and security of Let's Hash, we conducted a usability study with 179 freelance developers, who were asked to work on three short programming tasks in the context of password storage, password policies, and 2FA. Participants were split into three groups, one for each version of Let's Hash and one control group in which developers were allowed to use the resources they usually use during development.

The results show a vast improvement. Participants using Let's Hash were between 5 and 32 times as likely to achieve secure code than the control, depending on the task and version of Let's Hash. Post-hoc tests show that all improvements between Let's Hash and the control group are statistically significant. With these results, we believe that Let's Hash can offer a valuable contribution and help improve password security significantly.

## 2 Related Work

Authentication is a major part of security in IT, and it is susceptible to vulnerabilities in many ways. Attackers can gain unauthorized access to systems by manipulating or circumventing the authentication process, e.g., by guessing commonly chosen passwords [35, 53, 60], or through password leaks from databases [18, 29, 49]. End user focused research explored the difficulties that users have with security mechanisms in general [8, 57, 67], and specifically the authentication process like choosing and remembering passwords [30, 36, 63] or using alternative methods or second factors [17]. However, there is only limited knowledge of how to support software developers with secure programming [6, 33]. Recent work found that developers lack security expertise and often base their security decisions on misconceptions or outdated knowledge [6, 33, 42, 43]. But there already exist examples of APIs developed to support programmers with security, such as the Secure Socket API [45].

Let's Hash currently supports developers with password storage, password policies, and two-factor authentication, so we cover related work for each of these areas in the following.

### 2.1 Password Storage

For secure storage in a database, user passwords have to be salted and hashed [32]. Software developers, however, struggle with this task [7, 12, 29, 41–44, 68]. Previous studies showed that developers often search for programming code on the Internet to copy and paste it to their applications [4, 27, 41, 42, 44]. While Fischer et al. [27] and Acar et al. [4] found that this behavior can lead to functional but insecure software, in a password-storage study with developers of Naiakshina et al. [44], all participants who submitted secure programming code had copied and pasted it from the Internet. The authors analyzed the used websites in detail and found that participants adopted code from blog posts, tutorials, and Stack Overflow. In [41], Naiakshina et al. conducted a further study on password storage with developers. If participants submitted insecure solutions, the authors provided links to websites of the Open Web Application Security Project (OWASP) and the National Institute of Standards and Technology (NIST), where advice and programming code for secure user password storage was available. The results showed that guiding developers to appropriate information sources suitable to their programming use-cases can improve software security. However, 47% of participants did not find the appropriate security information without help from the authors.

### 2.2 Password Policies

To ensure that users choose passwords that are hard to guess for potential attackers, there are certain guidelines that are often implemented as requirements, commonly referred to as password policies [32, 46]. There have been multiple studies to examine the effect of enforcing such password policies [23, 56, 59, 61, 61, 62]. Requirements that target the passwords' composition, while common, do little to encourage users to pick better quality passwords. A combination of minimum length and minimum strength is more effective [59]. Password strength meters have also been investigated [23, 61, 62]. While stricter password policies can help users create better passwords, they also increase user frustration and reduce password retention [61]. Strength meters offering constructive feedback performed better [61]. Segreti et al. [56] investigated adaptive password policies, which aimed to increase password diversity by comparing new passwords to the existing password database, resulting in policies that changed as new passwords were added to the database. They found that this improved security at little cost to usability and that the additional feedback they provided on how to improve password security did not make much of a difference concerning usability [56]. To our knowledge, the implementation of password policies has not yet been investigated from a developer's point of view.

### 2.3 Two-Factor Authentication

Authentication can be made significantly more secure by adding different factors [52]. Yubico Security keys (Yubikeys) are an example of a hardware authentication device, which supports two-factor authentication (2FA) standards,

like the Universal 2nd Factor (U2F) and Fast Identity Online 2 (FIDO2) protocols [20]. Alam et al. [9] investigated possible pitfalls in implementing the new open source authentication standard, FIDO2, by evaluating discussions about it on Stack Overflow and assessing existing libraries and documentation. They found that documentation is currently not very usable, libraries implementing the standard are often both insecure and incomplete, and that developers have wrong mental models of implementing the standard and threats that FIDO2 protects against. The authors call for better support for developers to mitigate these issues.

# 3 Let's Hash

Let's Hash was loosely inspired by Let's Encrypt and Certbot [2]. The mission of Let's encrypt is to enable all admins to easily acquire and set up TLS certificates to combat the many sites that did not offer TLS at all or suffered from one of many misconfigurations. Our goal with Let's Hash is very similar. We want to enable all developers to easily integrate secure password storage, password policy enforcement, and two-factor authentication into their applications without falling prey to the many mistakes that can be made. The code snippets contained in Let's Hash are presented in a way that makes it easy for developers to copy and paste them into their projects since it was shown in previous studies that this is a common use-case (see Section 2). We designed Let's Hash according to websites like Stack Overflow [21] and blog posts [13], by presenting the code snippets divided by topic, but not split apart into single functions as is often seen in documentation [28]. Unlike NIST and OWASP, where theory and guidelines are detailed, Let's Hash offers a code-centric view, combining these guidelines with easily adaptable code.

Let's Hash currently supports password storage, policy enforcement, and two-factor authentication. In the following, we will highlight the most relevant aspects in these three areas.

## 3.1 Password Storage

There are a lot of different password hashing schemes (PHSs), which can be used in the context of user password storage (e.g., `MD5`, `SHA-1`, `SHA-2`, `PBKDF2`, `bcrypt`, `scrypt`, `Argon2`) [50]. We evaluated them for security and usability and included `Argon2id` as the most secure choice according to recent academic results [10, 14, 48, 50] and `bcrypt` as a more usable solution in contrast to `Argon2id` since it does not require manual adaption to specific hardware and is recommended by OWASP [48]. Iterations are configured based on the hashing algorithm. Currently, Let's Hash offers programming code snippets on secure user password storage in two programming languages, Python3 and Java.

## 3.2 Password Policies

NIST and OWASP both recommend policies that do not restrict password composition (i.e., allowing all kinds of characters but not enforcing a specific combination) and enforce a length of at least eight characters [32, 47]. Additionally, they advocate for using a strength checker, as is implemented by the library `zxcvbn` [66]. The German Federal Office for Information Security (BSI) advises users to choose their passwords according to popular composition rules - using upper- and lowercase letters and special characters [16]. Let's Hash offers a JavaScript solution to enforce BSI recommendations, ensuring a certain length and composition of a password. Additionally, there is a code fragment for a password strength checker using the aforementioned `zxcvbn`, as recommended by NIST and OWASP.

## 3.3 Two-Factor Authentication

Let's Hash offers a code fragment that generates a time-based token that serves as a one-time password. This token can be used as verification in conjunction with an app such as the Google Authenticator, which is a popular way to use 2FA [31]. Currently, programming code is provided in Python3.

## 3.4 Let's Hash Wizard

We have a two-component system. The code repository contains the code snippets for all the above tasks and sub-tasks, and a wizard assists developers in selecting and configuring the right snippets. We wanted to explore a design of Let's Hash that included a wizard-like user interface (UI)-element which required the user to interact with it.The wizard should present developers with the code that best fits their specific use-case by first taking them through a series of questions. These questions let developers pick secure implementation options according to the different recommendations provided by NIST, OWASP, and BSI. After stepping through the wizard, the appropriate code snippets are selected, configured, and presented to the developer ready to use.

Screenshots of the base version (LH) and the wizard (LH-W) and details on the code snippets are available on Github.[1] Let's Hash will also be released as an open-source project.

# 4 Methodology

To evaluate if using Let's Hash would increase security, we designed and ran an online study with freelance developers recruited from Freelancer.com as recommended by Naiakshina et al. [41]. All participants were asked to complete three short programming tasks on password storage, password policies, and 2FA. After task completion, we asked participants to fill out a survey assessing their experiences with the tasks and

[1] https://github.com/BeSecResearch/LetsHash-Supplemental

the information sources they used. The order of the tasks was randomized. We divided the participants into three groups:

- Group **LH:** Participants were asked to complete the programming tasks using version LH, the basic version of Let's Hash.
- Group **LH-W:** Participants were asked to complete the programming tasks using version LH-W of Let's Hash, the version with an added wizard for configuration.
- Group **C(ontrol):** Participants were asked to complete the programming tasks by using any information source they normally use when programming. These participants did not have access to Let's Hash.

## 4.1 Study Setup

For the study setup, we used the open-source tool Developer Observatory [58], which has been used in previous security studies about code development by Acar et al. [3, 7]. The tool allowed us to let participants work on programming tasks remotely. The participants could run and test the output of their code in a sandbox-like environment accessed via their browser. We provided them with function signatures and examples of expected results. This offered several advantages: First, participants did not have to download anything. They could directly access the consent form, the task description, the programming code, and the link to the follow-up survey in a browser of their choice. Second, participants did not need to spend time setting up their IDE. They could write and test the programming code within the tool. Third, we were able to log study data without the participants having to submit any of this data to us manually. In the following, we describe the programming tasks in detail.

## 4.2 Task Design

The exact task descriptions can be found in Appendix A. Examples of the study interface are available on Github.[2]

**Task 1 (T1) - Password hashing:** To keep the task as simple as possible, we only asked participants to implement a function for hashing and verifying passwords in Python. The solution required outputting a hash value and the correct verification of a given password. The task description as it was presented to participants can be found in Appendix A.

**Task 2 (T2) - Password policy:** We asked participants to implement a short JavaScript program checking a string for adherence to a given password policy. Since it is still a widely used practice in real life, we asked them to implement a policy that would enforce composition rules in addition to a minimum length. The solution of the task required an output that correctly breaks down a given password's adherence to the demanded policy. Appendix A contains the task description as it was presented to participants.

**Task 3 (T3) - Two-factor authentication:** We asked participants to implement a method that generates a time-based code that can be used as an authenticator. The solution to the task required an output containing the one-time code and its verification. The task description that was presented to study participants can be found in Appendix A.

## 4.3 Survey

In the survey, we asked participants of all groups about their experience with programming in general and the given topics in particular. They also indicated their perception of the difficulty of the programming tasks and were asked general demographic questions. Participants of groups LH and LH-W additionally had to answer the System Usability Scale (SUS) [37] for the version of Let's Hash that they used. Participants of the control group C were asked about the specific resources they used to solve the tasks and whether they were satisfied with them. The surveys for all the three groups can be found in Appendix B.

## 4.4 Usability Evaluation

In accord with ISO 9241, we define usability as encompassing effectiveness, efficiency, and user satisfaction [1].

**Effectiveness:** Every task submission was examined based on two criteria: functionality and security. To count as functional, the code submitted by the participants needed to run and produce an output that offered the information specified in the task description. Functionality was a prerequisite for security.

To determine whether the submitted code for Task 1 was secure, we adopted the security scale introduced by Naiakshina et al. in [43, 44] (see Appendix C). They used a scale of up to seven points for hashing and salting user passwords. We used the same scale, and only rated solutions as secure that reached at least 6 out of 7 points. We were strict in our evaluation because we were only interested in solutions that offered up-to-date security. This meant using a random salt and a key derivation function including an appropriate iteration count or a memory-hard function [32, 43]. We did not require 7 out of 7 points since the final point is for memory hardness which is not yet industry standard and we did not expect our participants to go beyond industry best practice. For Task 2, the code was rated as secure if the policy rules specified in the task description were correctly implemented without errors. The code for Task 3 was only rated as secure if the algorithm used to generate the second factor actually generated a time-based one-time code and used a salt that was randomly generated. The programming code was evaluated manually and independently by two computer science researchers. Differences were resolved through discussion.

[2] https://github.com/BeSecResearch/LetsHash-Supplemental

**Efficiency:** The number of clicks and time taken to solve a task are often used as efficiency variables in usability studies [26, 34, 38, 39, 55]. To evaluate how participants interacted with Let's Hash, we tracked the time (in seconds) they actively spent on the website and the clicks they needed to find the correct code fragments to use. We assumed that a long time and more clicks might affect the usability of the website [38, 39].

**User satisfaction and perceived usability:** We calculated the SUS score [37] for the two versions of Let's Hash. We used the SUS as one factor to compare the usability of the two versions of Let's Hash.

Additionally, we evaluated the answers to several open survey questions to gain insight into the participants' workflow as well as their general attitude towards IT security and Let's Hash. Since all answers were relatively short, and we were interested in specific themes, such as positive or negative attitudes towards Lets Hash, we used deductive thematic analysis to categorize and report on the participants' answers [15]. One researcher coded the entirety of the answers given, and a second researcher recoded them using the same codebook. Afterward, intercoder agreement was calculated per document. The minimum agreement was $\kappa = 0.76$, and the maximum agreement was $\kappa = 1$ (M=0.94). For the groups LH and LH-W, the questions were about the user experience with Let's Hash and how the website compares to other resources the developers would usually use to program. Participants of the control group C were asked to list the resources they used for the programming tasks. We categorized the answers into three types of resources: Stack Overflow, official documentation, and other, which included various blog posts and other resources.

**Usability and security:** To investigate the relationship between security and usability, we conducted Wilcoxon-Rank-Sum Tests for each of the different tasks, comparing submissions with errors (non-functional or insecure) with secure non-erroneous submissions, with respect to usability measures, such as SUS, time spent on Let's Hash and clicks. We corrected p-values using the Bonferroni-Holm procedure.

## 4.5 Error Analysis

We analyzed the types of errors we found in the participants' submissions to find out more about the kinds of errors that Let's Hash helps prevent and which ones still occur. To do this, we conducted a qualitative analysis. Each submitted solution for a task that was not both functional and secure was manually reviewed with regard to the types of errors it included. During the coding process, we assigned multiple different error types to a single submission, and ended up with a maximum of three different error types per submission. We estimated a lower limit of $\kappa = 0.84$ by ordering the error types per submission alphabetically and only taking into account the first one and an upper limit by counting the raters as in agreement, when they agreed over at least one of the assigned error types, which resulted in $\kappa = 0.91$.

We were also interested in errors occurring even though participants used Let's Hash. There are 53 non-functional or non-secure task solutions from groups LH and LH-W. We manually investigated whether these included copied code from Let's Hash. Both researchers judged 8 of them to be copied from Let's Hash, agreeing on 5 of them, and disagreeing on 3. The secure submissions in groups LH and LH-W were also tested on whether they had copied their code from Let's Hash or not. Due to the high amount of files for these cases this process was semi-automated. Details are in Appendix D.

All differences concerning the error types and the code-copying were resolved through further discussion, and full agreement was reached.

## 4.6 Hypotheses and Statistical Analysis

We were interested in the security of code developed with the help of Let's Hash. Additionally, we wanted to study Let's Hash's usability, encompassing efficiency and effectiveness [1]. Therefore, we examined four main hypotheses in our study: one on the security score between the groups LH and LH-W and the control group C, denoted by S(ecurity), and three concerning the differences between groups LH and LH-W, denoted by D(ifference). While we hoped that LH-W would improve both security and usability over LH, there is not enough theoretical foundation to justify one-tailed hypotheses, so all hypotheses were tested two-tailed at the standard p=.05 level throughout.

- H-S: The groups LH and LH-W, that are working with Let's Hash, produce code that is more **S**ecure than that produced by the control group C, that had no access to Let's Hash but could use any other source.
- H-D1: The System Usability Scale (SUS) **D**iffers between the two versions of Let's Hash.
- H-D2: There is a **D**ifference in the number of clicks needed to reach the desired code fragments using the two different versions of Let's Hash.
- H-D3: There is a **D**ifference in time that participants need to reach the desired code fragments using the two different versions of Let's Hash.

We used the freely available software Gnu R [51] for statistical analyses.

## 4.7 Pilot Study

We ran a pilot test before the main study to test the technical setup and ensure that we had correctly configured the Developer Observatory tool and the website Let's Hash. We recruited three participants. These were students who worked

Table 1: Demographics of 179 participants

| | | | |
|---|---|---|---|
| **Gender** | Male: 92% | Female: 8% | Prefer not to say: 0.6% |
| **Ages** | Mean: 28.6 | Median: 27 | SD: 7.5 |
| **Education and Occupation** | University Degree: 80% | Employed at company: 28% | |
| **Country of Origin** | India: 22% | Pakistan: 9% | Other: 69% |
| **Experience** (Programming language) | Python3: Mean: 3.2 Median: 3 SD: 2.3 | JavaScript: Mean: 4.3 Median: 4 SD: 3.3 | Overall: Mean: 6.4 Median: 5 SD: 5.0 |

as research assistants in security-related fields within computer science. All participants were male and aged between 20 and 40 years. The pilot study indicated that the setup of the Developer Observatory and Let's Hash worked as intended. The participants did not raise any serious issues, and we only made minor changes based on their feedback.

## 4.8 Power Analysis

We performed a power analysis based on our four main hypotheses to calculate the required sample size for this study. We used G*Power to perform two analyses [25], one for H-S, and one for H-D1, H-D2, and H-D3. We calculated a required sample size of at least 49 participants per group.

For H-S, we performed a Fisher's exact test, comparing the groups LH and LH-W against group C. For this power calculation, we merged groups LH and LH-W, since both versions of Let's Hash only differed in the existence of a wizard, but the code fragments were the same on both versions. We assumed that the code developed by LH and LH-W would be secure in 90% of the solved tasks, but the code for group C only in 60%. We based these percentages on the results of Acar et al., where a similar task on password storage was part of a user study with GitHub users [7]. Using these parameters, a desired error probability $\alpha$ of 0.05 and a desired power of 0.95 resulted in a sample size of 116 participants, 77 total for groups LH and LH-W, and 39 for group C.

With H-D1, H-D2, and H-D3, we aimed to figure out if the added wizard had a noticeable effect on the usability of Let's Hash. To compute a required sample size, we used the two-tailed Wilcoxon-Mann-Whitney test with two groups of equal sizes. Since Klug stated that "the average SUS score is 68 with a standard deviation of 12.5" [37], we set our baseline value accordingly for H-D1. We used mean values of 68 and 78 for the two groups, which would for instance improve the usability of Let's Hash from "Ok" to "Good" [11]. The standard deviation was set to 12.5 for both. This led to an effect size of 0.8. For H-D2 and H-D3, we used the same effect size for comparability since we were not aware of standardized measures for the number of clicks or the time spent on a website such as Let's Hash. Using this effect size and the same values for $\alpha$ and power as in our first analysis we calculated a required sample size between 28 and 49 participants per group, depending on the distribution, so we selected 49 to be on the safe side.

## 4.9 Participants

For our study, we needed developers with experience in Python and JavaScript. We used the support service of Freelancer.com for participant recruitment as suggested in [19,42]. All freelancers who finished the tasks and survey received 40€ for participation.

294 participants were invited to take part in our study to have enough participants to account for drop-outs and other issues. All 294 participants provided informed consent. Of these, 239 completed the tasks and the survey, 55 quit the study midway. The data of 60 participants was not considered for analysis for varying reasons. 31 of them were removed because we found inconsistencies in the recorded tracking, which showed that multiple participants had access to the wrong version of the website Let's Hash or even to both versions. Some of these inconsistencies were caused by a bug in the Developer Observatory tool, which was reported by us upon discovery. 15 participants could not be tracked by Let's Hash at all and had to be disregarded for that reason, 13 were excluded due to technical problems, 9 leading to data loss and 4 with incorrect condition assignment, and 1 participant was removed for speeding through the survey and giving nonsensical answers. Overall, 179 participants produced valid results, 58 in group LH, 57 in group LH-W, and 64 in group C exceeding our 49 target.

The majority of the participants were male (92%), while only 8% were female, and one person preferred not to disclose their gender, which is fairly typical for these platforms. They reported ages between 18 and 70 years (M=28.6, SD=7.5) and between 0 and 30 years of programming experience overall (M=6.4, SD=5.0), with slightly less experience in Python3 and JavaScript. Most participants were not from countries where English is the only primary language, with Indians (22%) and Pakistanis (9%) representing the largest groups of nationalities in our sample. The remaining 69% of participants were from a variety of different countries, none of which amounted to more than 4%. Further demographics information about the participants is in Table 1.

Table 2: Distribution of correct solutions by task and group

| Task | | LH | LH-W | C |
|---|---|---|---|---|
| 1 | Functionality | 97% | 95% | 81% |
| | Security | 93% | 82% | 33% |
| 2 | Functionality | 95% | 91% | 70% |
| | Security | 91% | 75% | 36% |
| 3 | Functionality | 97% | 88% | 59% |
| | Security | 86% | 79% | 16% |

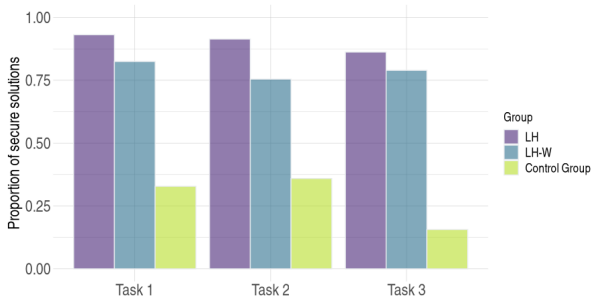


Figure 1: Secure solutions, divided by task and group.

## 5 Limitations

As usual for usability studies, several limitations have to be considered in the context of this study. Firstly, we did not conduct the study in a lab setting. Due to requiring a high number of software developers and the ongoing COVID-19 pandemic, we conducted an online study. Consequently, we had less control and were not able to track participants' processes when working on the tasks.

Secondly, we used Freelancer.com for study recruitment, which limited the pool of possible participants to people registered on this platform. The vast majority of the recruited freelancers were not native English speakers, which might have lead to misunderstandings due to a language barrier. However, real-world projects are often outsourced to freelancers under similar conditions.

Thirdly, due to our study setting, our tasks were rather short, did not include finding the resource and we provided implementation stubs to prevent using developers' time unduly. Thus, participants might not have put as much effort into the task, or have applied different priorities than in a real task. However, results from a lab study have shown to be comparable to those in a field study in the context of password storage [41, 42]. Our work provides the basis for further investigation.

Finally, the tracking of time participants spent on Let's Hash and the number of clicks they needed to achieve their goal was not as accurate as we would have wished due to the remote nature of the study. We could not capture the actual screen and had to attempt to log the time on the server-side. In some cases during testing, clicks were either not registered or counted twice. Additionally, we only recorded the interactive time spent on Let's Hash. If participants stopped scrolling, clicking, or moving their mouse for more than a minute, the timer stopped increasing. We wanted to avoid recording time during which users had the website open in a tab, but they were not actually looking at it, for example, because it was minimized. Considering this, these values should be taken as a best-effort approximation.

## 6 Ethics

This study was conducted in Germany and is compliant with the EU General Data Protection Regulation, a directive concerning the collection and storage of data. It also has Institutional Review Board (IRB) approval. All data was collected anonymously. Participants were asked to agree to a consent form that informed them about the study procedure and their rights, for example, to withdraw at any point during the study. Furthermore, we complied with the privacy policies of Freelancer.com, which meant we were not permitted to ask participants for their email addresses for further research or to inform them about study results.

## 7 Results

For evaluation, we compared the results of participants using Let's Hash and information sources of their own choice, as well as the two versions of Let's Hash.

### 7.1 Participants' Submissions

The participants who finished the study self-reported taking a median time of one hour to solve the programming tasks. Table 2 shows an overview of the evaluation of participants' submissions concerning functionality and security. We only rated security if the programming code was functional and included only functional solutions in our statistical tests comparing security. All participants combined produced functional solutions for Task 1 in 91%, Task 2 in 85% and Task 3 in 80% of the cases, and secure solutions for Task 1 in 68%, Task 2 in 66% and Task 3 in 59% of the cases.

#### 7.1.1 Functionality

Of all the participants who produced valid data, 168 submitted programming code that we considered functional for at

Table 3: Overview over the results of the 3-way FET for H-S

| Hypothesis | IV | DV | *p*-value | *cor* - *p*-value |
|---|---|---|---|---|
| H-S: T1 | LH vs. LH-W vs. C | Achieved security | <0.001* | <0.001* |
| H-S: T2 | LH vs. LH-W vs. C | Achieved security | <0.001* | <0.001* |
| H-S: T3 | LH vs. LH-W vs. C | Achieved security | <0.001* | <0.001* |

T1: Task on secure password storage, T2: Task on password policy, T3: Task on 2FA;
IV: Independent Variable; DV: Dependent Variable; FET: Fisher's Exact test
*cor* - *p*-value: *p*-value, Bonferroni-Holm corrected; tests marked with *: statistically significant.

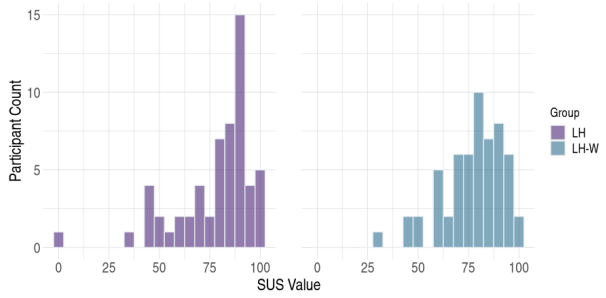


Figure 2: Distribution of SUS values.
One bar covers a range of n+5 points.

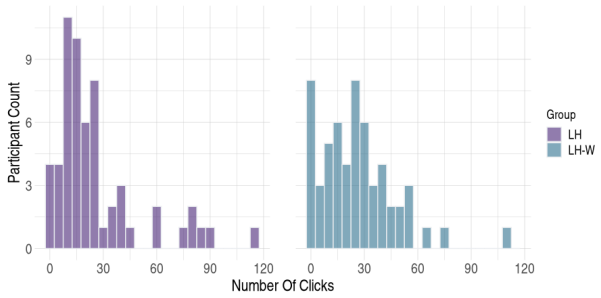


Figure 3: Distribution of counted clicks.
One bar covers a range of n+5 clicks.

least one of the tasks. The task with the most functional solutions overall was Task 1. For this task, 162 participants submitted a functional solution. Of these, 157 participants indicated that they had previous experience with storing passwords in a database. For Task 2, 152 participants submitted a functional solution, and 143 participants indicated they had previous experience with implementing password policies in the context of a login form. The task that produced the least amount of functional solutions was Task 3. 144 participants submitted a functional solution for Task 3. Of these, 96 participants indicated that they had previous experience with implementing 2FA. Compared to the two other tasks, 2FA seems less frequently demanded among the population on Freelancer.com.

#### 7.1.2 Security

The task with the most secure solutions overall was Task 1 with a total of 122 secure solutions, followed by Task 2 with 119 secure solutions. Task 3 had the least amount of secure solutions with only 105.

### 7.2 Hypotheses

#### 7.2.1 Security

Figure 1 shows the proportions of secure solutions, divided by groups and tasks. We conducted separate 3-way Fisher's Exact tests for each programming task to test H-S. We corrected p-values for multiple testing using the Bonferroni-Holm-correction since we conducted three tests for this hypothesis. All p-values for the main analyses, including the corrected ones, are below 0.001, as can be seen in Table 3. Our analyses indicated that for all of the tasks, there was a significant difference with respect to security between the three groups. We then conducted post-hoc 2-way Fisher's Exact tests to compare the groups individually (see Table 5) and included these in our correction. We found that for all of the tasks, the solutions achieved with the help of Let's Hash were significantly more secure than those achieved with resources of the participants' own choosing. This shows that the primary goal of our work was achieved. Let's Hash significantly increases the odds of developers creating secure solutions by a large margin.

#### 7.2.2 Usability

To compare the usability of the two Let's Hash versions, we evaluated the SUS, the number of clicks, and the time spent actively on each version of the website. Since neither of the groups LH or LH-W had a normal distribution in SUS values, time spent, or amount of clicks used on the website Let's Hash, we performed Wilcoxon-Rank-Sum tests. The results of our statistical analyses of H-D1, H-D2, and H-D3 are available in Table 4.

For H-D1, we did not find a significant difference in usability as measured by SUS between the version with (M=78.6, median=80, SD=15.1) and the version without a wizard (M=79.1, median=87.5, SD=19.3). In general, both versions of Let's Hash achieved results that were close to being con-

Table 4: Overview over the results of the WRS tests for H-D1, H-D2 and H-D3

| Hypothesis | IV | DV | $\mathcal{W}$ | $r$ | $p$-value |
|---|---|---|---|---|---|
| H-D1 | LH vs. LH-W | Achieved SUS | 1500 | 0.08 | 0.3926 |
| H-D2 | LH vs. LH-W | Amount of clicks | 1804 | 0.08 | 0.3996 |
| H-D3 | LH vs. LH-W | Time spent on website | 1329 | 0.17 | 0.0704 |

IV: Independent Variable; DV: Dependent Variable; WRS: Wilcoxon-Rank-Sum test;
$\mathcal{W}$: Wilcoxon-$\mathcal{W}$; $r$: Effect size (Pearson's $r$)

Table 5: Overview over the results of the post-hoc 2-way FETs for H-S

| IV (Group) | Task | DV | OR | CI | $p$-value | *cor* - $p$-value |
|---|---|---|---|---|---|---|
| LH vs. LH-W | Password Storage | Achieved security | 0.252 | [0.024, 1.408] | 0.09 | 0.19 |
| LH vs. LH-W | Password Policy | Achieved security | 0.183 | [0.018, 0.95] | 0.03* | 0.08 |
| LH vs. LH-W | 2-Factor Authentication | Achieved security | 1.079 | [0.255, 4,798] | 1 | 1 |
| C vs. LH-W | Password Storage | Achieved security | 9.668 | [3.483, 30.399] | <0.001* | <0.001* |
| C vs. LH-W | Password Policy | Achieved security | 4.115 | [1.521, 11.958] | 0.002* | 0.009* |
| C vs. LH-W | 2-Factor Authentication | Achieved security | 23.886 | [6.985, 100.032] | <0.001* | <0.001* |
| C vs. LH | Password Storage | Achieved security | 38.372 | [8.543, 359.027] | <0.001* | <0.001* |
| C vs. LH | Password Policy | Achieved security | 22.456 | [4.873, 212.115] | <0.001* | <0.001* |
| C vs. LH | 2-Factor Authentication | Achieved security | 22.208 | [6.884, 84.436] | <0.001* | <0.001* |

FET: Fisher's Exact test; IV: Independent Variable; DV: Dependent Variable; O.R.: Odds ratio; C.I.: Confidence interval
*cor* - $p$-value: $p$-value, Bonferroni-Holm corrected, including nine post-hoc tests and three main 3-way analyses; tests marked with *: statistically significant

sidered excellent in usability [11, 54]. Figure 2 shows the distribution of SUS values for both versions of Let's Hash. The majority of ratings designate the usability of both versions as good (>71), although there was slightly more variance in ratings for group LH, which used the website without a wizard.

For hypothesis H-D2, we also did not find a significant difference in clicks needed between group LH (M=25.6, median=17.5, SD=25.1) and group LH-W (M=26.2, median=24, SD=21.5). Figure 3 shows the distribution of the counted clicks. Again, the distributions are roughly similar, but more participants issued very few clicks in group LH-W, and more participants issued high numbers of clicks (>60) in group LH.

To test H-D3, we compared the time actively spent on the website. Participants of group LH spent slightly more time on their version of the website (M=233.8, median=179, SD=200.5) than those in group LH-W (M=176.3, median=115, SD=156.4), but the difference was not significant. The time participants spent on Let's Hash is available in Figure 4. Participants spent a median of fewer than 3 minutes on version LH and fewer than 2 minutes on version LH-W.

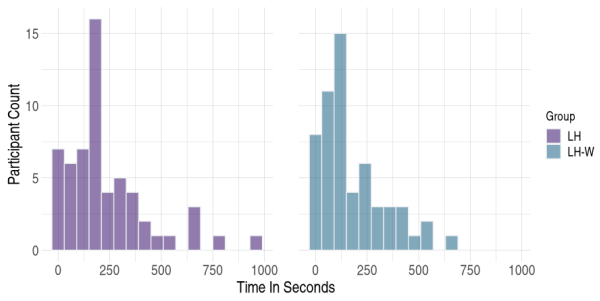


Figure 4: Distribution of time spent on Let's Hash.
One bar covers a range of n+60 seconds.

#### 7.2.3 Hypothesis Takeaways

The main takeaway of the hypotheses analysis can be summarized as follows: Using either version of Let's Hash as a resource during code development has a large and significant positive effect on the security of the developed code.

To our surprise, the usability of the two versions was rated almost identically, and thus, there were no statistically significant differences between the two, implying that the wizard did not improve usability as we had expected. Even more surprisingly, LH has higher security odds than the LH-W. While the difference was not statistically significant, we still find it interesting and discuss it further in the following section. For now, we conclude that the plain Let's Hash resource improves the odds for a secure solution between 17 and 32 times compared to the control and does not have any downsides compared to LH-W, so the extra effort for the wizard does not seem justified or necessary. Although further research into the reasons why is recommendable.

### 7.3 Error Analysis

Despite the excellent results, some participants created insecure code despite using Let's Hash. This section analyzes these errors and examines the usability judgments of participants making errors despite using Let's Hash. We found eight

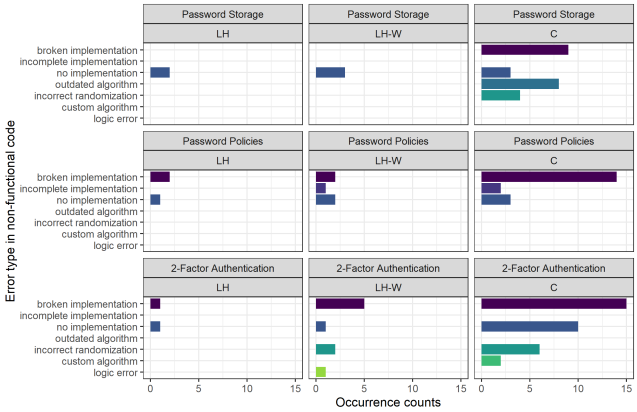


Figure 5: Types of errors in non-functional code, divided by task and group.

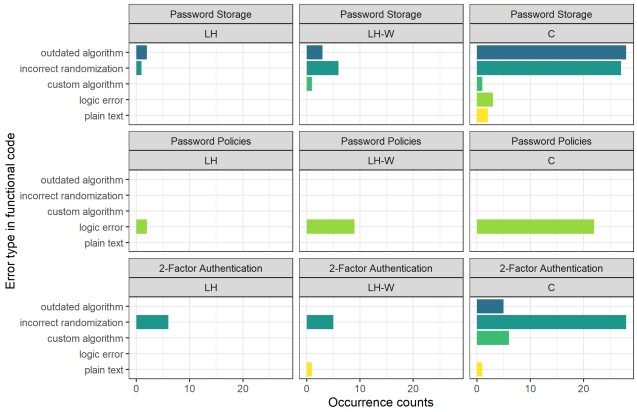


Figure 6: Types of security errors in functional code, divided by task and group.

different types of errors in the participants' submissions, three related to functionality, and five types concerning security. Even for non-functional submissions, we also documented security errors.

Control group submissions proportionally more often had multiple error types, which led to them lacking in security and/or functionality. This is in addition to boasting more errors overall, as has already been established and shows that the insecurity and/or non-functionality of code was often due to multiple types of errors and not merely one cause.

The errors participants made that would lead to their code being non-functional were categorized as 'no implementation' when there was no attempt at a solution, 'incomplete implementation' when the participants did not finish the task and 'broken implementation' when the code contained severe programming errors. As can be seen in Figure 5, 'broken implementation' was the most common error, with 11 cases overall in LH and LH-W and 38 cases in group C. We included 'no implementation' as a category under the assumption that participants were likely not unwilling to solve the task, but instead overwhelmed. The security errors were 'plain text', 'outdated algorithms' and 'use of custom algorithms', which all refer to the cryptographic algorithms implemented by participants in tasks 1 and 3. For example, some participants used the outdated md5 to hash the password in Task 1, which we classified accordingly as 'outdated algorithms'. Another type of error concerning these two tasks is 'incorrect randomization', when the salt or shared secret was either too short, or not sufficiently randomized to be considered secure. Figure 6 shows that this was the most common security error type for the functional cryptographic tasks in all groups, with 18 overall occurrences in groups LH and LH-W and 55 in group C. One instance of this error was a participant who used their own first name as the shared secret in Task 3. None of these errors apply to Task 2 since there was no cryptography involved in this task. Instead, the security error consisted of participants implementing a function that appeared to deliver the requested results but was not consistent. This leads to issues such as passwords being erroneously classified as adhering to policy. We refer to these errors as 'logic error'.

### 7.3.1 Non-functional Submissions

The frequency of different types of errors for those submissions which were classified as non-functional is depicted in Figure 5. It shows that non-functional solutions also suffer from security errors, in addition to purely functionality-related errors. This is especially common in the control group. In general, a wider range of different errors occurred in the control group compared to the two Let's Hash groups. The most striking example was in the Password Storage task, where LH and LH-W participants' submissions were non-functional because they did not implement anything, but group C submissions exhibited a wide range of functional and security errors.

### 7.3.2 Functional, but Non-secure Submissions

The frequency of different error types for submissions which were functional, but insecure is shown in Figure 6. Like for non-functional solutions, there were fewer different types of errors in the Let's Hash groups. For example, for the password storage task, there were five different types of security errors in the control group, but only two types of security in the LH and three types in the LH-W group. Some types of errors, the plain text and logic errors, were more common in functional solutions than non-functional solutions.

#### 7.3.3 Errors Despite Using Let's Hash

In groups LH and LH-W, most errors occurred when participants did not use the code provided by Let's Hash. Of 53 tasks where participants submitted erroneous solutions, 45 did not use copied code from Let's Hash. In contrast, of 292 correct tasks in the groups LH and LH-W, only 19 were not copied. Overall, the most common task in which participants did not choose to use copied code was Task 2. This may be because the code had to be adapted more than in the other two tasks, since participants would have had to change the definition of a variable. Common mistakes that participants made when they did not copy code from Let's Hash were using outdated algorithms like md5 (3 cases) or hard-coding the salt (12 cases). One participant submitted a solution for Task 3 which did not contain a cryptographic algorithm at all. There were only eight submissions for tasks where participants copied code from Let's Hash but still created an insecure solution. Four of those had functionality errors which we classified as broken implementation. These errors mainly were caused by participants not adapting the code correctly from the website, or introducing faulty syntax, like indentation errors. The remaining four occurrences all fall into LH-W. Three of the errors were in the password storage task, and all included incorrect randomization. Two of those participants had removed the generation of the salt from the function hash_password() and instead used a static variable, and one of the participants lowered the amount of rounds for the salt's generation to 4 instead of the recommended minimum of 16. The remaining error occurred in the password policies task and was a logic error, specifically in the implementation of the function composition(), which would allow a password without any special characters to pass the check, violating the policy requested in the task description. This error was introduced because the participant either removed or did not copy a part of the function.

#### 7.3.4 Usability and Errors

We found significant differences in usability between tasks with errors and those without for Task 1 concerning clicks, and for Task 3 concerning SUS, time (in seconds) and clicks. Participants with erroneous submissions issued significantly fewer clicks on Let's Hash both for Task 1 (M=9.4, SD=8.5) and Task 3 (M=9.1, SD=10.4) than participants with secure submissions (Task 1: M=28.2, SD=23.8; Task 3: M=29.4, SD=23.7), W=1116.5, corrected-p=.004 for Task 1, W=1557.5, corrected-p<.001 for Task 3. Participants who submitted wrong solutions also spent significantly less time on Let's Hash for Task 3 (M=93.5, SD=76.9) than those with secure solutions (M=229, SD=189), W=1418, corrected p=.004. Finally, participants with secure solutions rated Let's Hash with as significantly more usable (SUS score M=81.3, SD=16.0 ) than those with errors in their submissions (M=67.3, SD=19.1), W=1393, corrected p=.006. This suggests that the participants who made mistakes despite having Let's Hash at hand may have abandoned this resource early on in their coding process before being able to solve their problem. The full results of this analysis are available on Github.[3]

### 7.4 Participants' Feedback on Resources

In general, feedback on Let's Hash as a resource for code development was positive. 57 participants gave detailed feedback, and of those, 40 participants reported that they found the website easy to use, and 28 participants said they found it pleasant, enjoyed the UI, and wished to use it again. Requested changes included the addition of more languages (both programming languages and spoken languages), tutorials on how to use the code, the ability to run the code directly on Let's Hash in a sandbox-like environment, and improvements to the UI. The most requested change was additional information on the presented code and security-related challenges, which was mentioned by 34 participants. This request suggests that in their usual workflow, most developers do look for at least some background information when incorporating code found online into their work. Furthermore, 56 of the 115 participants who worked with Let's Hash reported that they found Let's Hash to be generally easier to use than their usual resource, citing reasons such as easy navigation and a well-structured presentation of the code. One participant mentioned that he would trust Let's Hash more than his usual resource in terms of security since it is "not a forum post."

68 participants from groups LH and LH-W indicated that the main resource they would usually use is Stack Overflow. 41 said they would use the official documentation, but almost none of them cited other resources and those that did mostly indicated they would usually "search on google."

Participants of group C also most commonly mentioned Stack Overflow and official documentation. Both resources were mentioned by more than 20 of the 65 participants (>30%), and 18 of them (28%) indicated that Stack Overflow was their main resource. Other websites that were mentioned in group C were various blogs and some online schools like W3Schools [65] or Vitosh Academy [64].

## 8 Discussion

We found that Let's Hash significantly improved the security of our participants' code. Although all participants were asked to solve the three authentication tasks securely, most secure solutions were submitted by participants using Let's Hash – regardless of which Let's Hash version they were using. While for both groups LH and LH-W, the submitted solutions were secure in at least 82% of the cases for Task 1, 75% for Task 2 and 79% for Task 3, 72% of participants in the

[3] https://github.com/BeSecResearch/LetsHash-Supplemental

control group submitted insecure programming code, with 33% secure solutions for Task 1, 36% for Task 2, and 16% for Task 3, which is alarming. These results suggest a large positive effect of Let's Hash on software security.

We also compared the efficiency and perceived usability of the two Let's Hash versions. With the configuration wizard, we wanted to take a burden off developers and provide them with a better overview of current recommended security practices for the three security-sensitive authentication tasks. However, we did not find a significant difference between the two Let's Hash versions concerning participants' perceived usability, which we measured with a SUS score. The usability was fairly high for both versions. So it seems participants were satisfied with using either Let's Hash version. In the follow-up survey, 24% of participants also reported that they felt supported by Let's Hash and would use it again. Most importantly, participants indicated that they trusted Let's Hash more than their usual resources. The fact that trust can impact the chosen resources and thus indirectly affect software security was already reported in [42]. We believe that trust and a high measurement of perceived usability are key factors for the successful establishment of Let's Hash.

Furthermore, we did not find a significant difference in the number of clicks and the time participants needed to solve the tasks with either version of Let's Hash. With an average of fewer than 26 clicks and 3 minutes to solve the tasks with either Let's Hash version, participants completed the tasks in a short time with little effort. With a success rate in terms of functionality of at least 95% for Task 1, 91% for Task 2, and 88% for Task 3, almost none of the participants gave up.

That there was no significant difference in clicks or time between the Let's Hash versions is especially interesting since participants using version LH-W had to interact with a wizard and decide between different requirements for the tasks. This wizard requires some development and maintenance effort. The fact that we did not observe a significant difference in the effectiveness, efficiency, and perceived usability between the two Let's Hash versions suggests omitting the wizard might be prudent. Then, the secure code snippets will be directly presented to developers, as they were in version LH. In either case, users can simply copy and paste the presented code into their projects. Having a central resource that is known to contain up-to-date code may help to mitigate the difficulties that developers have when looking for and assessing resources [4, 27]. We hope that by expanding Let's Hash, its relevance will increase over time. One such addition could be an implementation of a Single-Sign On (SSO). We plan to publish Let's Hash and build an open-source community for researchers and developers.

## 9 Conclusion

Previous work showed that developers struggle to adhere to security best practices. Programming resources aimed at helping developers work often are either complex, hard to understand and to use but secure, or easy to use but outdated and poorly maintained concerning security. To improve software security, we developed Let's Hash, a resource to support developers in implementing the security-critical authentication tasks: user password storage in a database, password policies, and 2FA.

The difference in security achieved with either version of Let's Hash compared to the developers' usual resources was highly significant. We further found that the two versions of Let's Hash did not differ significantly in either SUS score, time spent, or the number of clicks needed. The participants' perceived usability of both Let's Hash versions was excellent, and the participants' feedback was highly positive.

Our results indicated that Let's Hash has a great potential to improve the security of code that developers produce while also decreasing the effort needed. Consequently, we plan to deploy Let's Hash as a resource for developers and researchers. Future efforts could include incorporating additional topics, like SSO, or programming languages and more background information. Also, it might be helpful to explore how to best highlight security-critical parts of the code that should not be altered to mitigate some of the errors participants made while using Let's Hash. To keep Let's Hash well maintained, we will be releasing it as an open-source project on GitHub, and we hope to build a community.

## Acknowledgments

This work was partially funded by the Werner Siemens Foundation, and the ERC Grant 678341: Frontiers of Usable Security. The authors would like to thank Martin Welsch, Manfred Paul and Ben Swierzy for their help in developing and evaluating Let's Hash. We thank our anonymous reviewers and shepherd for helping us improve our paper.

## References

[1] Ergonomics of human-system interaction - part 11: Usability: Definitions and concepts. Technical Report ISO 9241-11:2018, March 2018.

[2] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, et al. Let's encrypt: an automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2473–2487, 2019.

[3] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. Comparing the usability of cryptographic apis. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171, 2017.

[4] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305. IEEE, 2016.

[5] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L Mazurek, and Christian Stransky. How internet resources might be helping you develop faster but less securely. *IEEE Security & Privacy*, 15(2):50–60, 2017.

[6] Yasemin Acar, Sascha Fahl, and Michelle L Mazurek. You are not your developer, either: A research agenda for usable security and privacy research beyond end users. In *2016 IEEE Cybersecurity Development (SecDev)*, pages 3–8. IEEE, 2016.

[7] Yasemin Acar, Christian Stransky, Dominik Wermke, Michelle L Mazurek, and Sascha Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95, 2017.

[8] Anne Adams and Martina Angela Sasse. Users are not the enemy. *Commun. ACM*, 42(12):40–46, December 1999.

[9] Aftab Alam, Katharina Krombholz, and Sven Bugiel. Poster: Let history not repeat itself (this time) – tackling webauthn developer issues early on. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 2669–2671, New York, NY, USA, 2019. Association for Computing Machinery.

[10] Joël Alwen and Jeremiah Blocki. Towards practical attacks on argon2i and balloon hashing. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 142–157, 2017.

[11] Aaron Bangor, Philip Kortum, and James Miller. Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123, 2009.

[12] Jason Bau, Frank Wang, Elie Bursztein, Patrick Mutchler, and John C Mitchell. Vulnerability factors in new web applications: Audit tools, developer selection & languages. *Stanford, Tech. Rep*, 2012.

[13] Ruan Bekker. Salt and hash example using python with bcrypt on alpine, 2018. Last retrieved April 30, 2021 from https://blog.ruanbekker.com/blog/2018/07/04/salt-\and-hash-example-using-python-with-bcrypt-\on-alpine/.

[14] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. On the economics of offline password cracking. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 853–871, 2018.

[15] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[16] BSI. Bundesamt für sicherheit in der informationstechnik, 2021. Last retrieved April 21, 2021 from https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Cyber-Sicherheitsempfehlungen/Accountschutz/Sichere-Passwoerter-erstellen/sichere-passwoerter-erstellen_node.html.

[17] Jessica Colnago, Summer Devlin, Maggie Oates, Chelse Swoopes, Lujo Bauer, Lorrie Cranor, and Nicolas Christin. "it's not actually that horrible" exploring adoption of two-factor authentication at a university. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–11, 2018.

[18] Nik Cubrilovic. Rockyou hack: From bad to worse, 2009. Last retrieved February 19, 2021 from https://techcrunch.com/2009/12/14/rockyou\-hack-security-myspace-facebook-passwords/.

[19] Anastasia Danilova, Alena Naiakshina, Johanna Deuter, and Matthew Smith. Replication: On the ecological validity of online security developer studies: Exploring deception in a password-storage study with freelancers. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 165–183. USENIX Association, August 2020.

[20] Sanchari Das, Andrew Dingman, and L Jean Camp. Why johnny doesn't use two factor a two-phase usability study of the fido u2f security key. In *International Conference on Financial Cryptography and Data Security*, pages 160–179. Springer, 2018.

[21] C. Dutrow and M. Amery. Salt and hash a password in python, 2019. Last retrieved April 30, 2021 from https://stackoverflow.com/questions/9594125/salt-and-hash-a-password-in-python.

[22] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 73–84. ACM, 2013.

[23] Serge Egelman, Andreas Sotirakopoulos, Ildar Muslukhov, Konstantin Beznosov, and Cormac Herley. Does

my password go up to eleven? the impact of password meters on password selection. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 2379–2388, New York, NY, USA, 2013. Association for Computing Machinery.

[24] Sascha Fahl, Marian Harbach, Henning Perl, Markus Koetter, and Matthew Smith. Rethinking ssl development in an appified world. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 49–60, 2013.

[25] Franz Faul, Edgar Erdfelder, Albert-Georg Lang, and Axel Buchner. G* power 3: A flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behavior research methods*, 39(2):175–191, 2007.

[26] Juan M Ferreira, Silvia T Acuna, Oscar Dieste, Sira Vegas, Adrian Santos, Francy Rodriguez, and Natalia Juristo. Impact of usability mechanisms: An experiment on efficiency, effectiveness and user satisfaction. *Information and Software Technology*, 117:106195, 2020.

[27] F. Fischer, K. Böttinger, H. Xiao, C. Stransky, Y. Acar, M. Backes, and S. Fahl. Stack overflow considered harmful? the impact of copy paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 121–136, 2017.

[28] Flask. Flask-bcrypt, 2020. Last retrieved April 30, 2021 from https://flask-bcrypt.readthedocs.io/en/latest/.

[29] Dinei Florêncio, Cormac Herley, and Paul C Van Oorschot. An administrator's guide to internet password research. In *28th Large Installation System Administration Conference (LISA14)*, pages 44–61, 2014.

[30] Alain Forget, Sonia Chiasson, and Robert Biddle. Helping users create better passwords: Is this the right approach? In *Proceedings of the 3rd Symposium on Usable Privacy and Security*, pages 151–152, 2007.

[31] Google. Google authenticator, 2020. Last retrieved March 23, 2020 from https://github.com/google/google-authenticator.

[32] Paul A Grassi, James L Fenton, EM Newton, RA Perlner, AR Regenscheid, WE Burr, JP Richer, NB Lefkovitz, JM Danker, Yee-Yin Choong, et al. Nist special publication 800-63b: Digital identity guidelines. *Enrollment and Identity Proofing Requirements. url: https://pages. nist. gov/800-63-3/sp800-63a. html*, 2017.

[33] Matthew Green and Matthew Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[34] Aleksander Groth and Daniel Haslwanter. Efficiency, effectiveness, and satisfaction of responsive mobile tourism websites: a mobile usability study. *Information Technology & Tourism*, 16(2):201–228, 2016.

[35] Troy Hunt. Have i been pwned. *Last retrieved*, 23, 2019.

[36] Saranga Komanduri is a Phd. Helping users create better passwords. 2012.

[37] Brandy Klug. An overview of the system usability scale in library website and system usability testing. *Weave: Journal of Library User Experience*, 1(6), 2017.

[38] Philip Kortum and Claudia Ziegler Acemyan. The relationship between user mouse-based performance and subjective usability assessments. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 60, pages 1174–1178. SAGE Publications Sage CA: Los Angeles, CA, 2016.

[39] Gitte Lindgaard and Cathy Dudek. *User Satisfaction, Aesthetics and Usability*, pages 231–246. Springer US, Boston, MA, 2002.

[40] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. Jumping through hoops: Why do java developers struggle with cryptography apis? In *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, pages 935–946, New York, NY, USA, 2016. ACM.

[41] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. On conducting security developer studies with cs students: Examining a password-storage study with cs students, freelancers, and company developers. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[42] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. "if you want, i can store the encrypted password" a password-storage field study with freelance developers. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2019.

[43] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, Marco Herzog, Sergej Dechand, and Matthew Smith. Why do developers get password storage wrong? a qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 311–328, 2017.

[44] Alena Naiakshina, Anastasia Danilova, Christian Tiefenau, and Matthew Smith. Deception task design in developer password studies: Exploring a student sample. In *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, pages 297–313, 2018.

[45] Mark O'Neill, Scott Heidbrink, Jordan Whitehead, Tanner Perdue, Luke Dickinson, Torstein Collett, Nick Bonner, Kent Seamons, and Daniel Zappala. The secure socket {API}:{TLS} as an operating system service. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 799–816, 2018.

[46] OWASP. Open web application security project, 2021. Last retrieved April 26, 2021 from https://owasp.org/.

[47] OWASP. Owasp authentication cheat sheet, 2021. Last retrieved February 20, 2021 from https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html.

[48] OWASP. Owasp password storage cheat sheet, 2021. Last retrieved February 20, 2021 from https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html.

[49] Sarah Perez. Recently confirmed myspace hack could be the largest yet, 2016. Last retrieved February 19, 2021 from https://techcrunch.com/2016/05/31/recently-confirmed-myspace-hack-could-be-\the-largest-yet/.

[50] PHC. Password hashing competition, 2019. Last retrieved February 20, 2021 from https://www.password-hashing.net/.

[51] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2020. https://www.R-project.org/.

[52] Joshua Reynolds, Trevor Smith, Ken Reese, Luke Dickinson, Scott Ruoti, and Kent Seamons. A tale of two studies: The best and worst of yubikey usability. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 872–888. IEEE, 2018.

[53] RockIt. The most common passwords 2020, 2020. Last retrieved February 19, 2021 from https://rockit.cloud/2020/03/18/the-most-commonly-used-password-in-2020-is/.

[54] Jeff Sauro. 5 ways to interpret a sus score, 2018. Last retrieved April 25, 2021 from https://measuringu.com/interpret-sus-score/.

[55] Jeff Sauro and James R. Lewis. Average task times in usability tests: What to report? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 2347–2350, New York, NY, USA, 2010. Association for Computing Machinery. https://doi.org/10.1145/1753326.1753679.

[56] Sean M. Segreti, William Melicher, Saranga Komanduri, Darya Melicher, Richard Shay, Blase Ur, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Michelle L. Mazurek. Diversify to survive: Making passwords stronger with adaptive policies. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 1–12, Santa Clara, CA, July 2017. USENIX Association.

[57] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *Symposium On Usable Privacy and Security*, pages 3–4. ACM, 2006.

[58] Christian Stransky, Yasemin Acar, Duc Cuong Nguyen, Dominik Wermke, Doowon Kim, Elissa M. Redmiles, Michael Backes, Simson Garfinkel, Michelle L. Mazurek, and Sascha 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, August 2017. USENIX Association.

[59] Joshua Tan, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Practical recommendations for stronger, more usable passwords combining minimum-strength, minimum-length, and blocklist requirements. 2020.

[60] Dan U. Passwords, passwords everywhere, 2019. Last retrieved February 19, 2021 from https://www.ncsc.gov.uk/blog-post/passwords-passwords-everywhere.

[61] Blase Ur, Felicia Alfieri, Maung Aung, Lujo Bauer, Nicolas Christin, Jessica Colnago, Lorrie Faith Cranor, Henry Dixon, Pardis Emami Naeini, Hana Habib, et al. Design and evaluation of a data-driven password meter. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3775–3786, 2017.

[62] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 65–80, Bellevue, WA, August 2012. USENIX Association.

[63] Blase Ur, Fumiko Noma, Jonathan Bees, Sean M Segreti, Richard Shay, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. " I Added'!'at the End to Make It Secure": Observing Password Creation in the Lab. In *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*, pages 123–140, 2015.

[64] Vitosh. Vitosh academy, 2021. Last retrieved April 25, 2021 from https://www.vitoshacademy.com/.

[65] W3Schools. W3schools online web tutorials, 2021. Last retrieved April 25, 2021 from https://www.w3schools.com/.

[66] Daniel Lowe Wheeler. zxcvbn: Low-budget password strength estimation. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 157–173, 2016.

[67] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *USENIX Security Symposium*, volume 348, pages 169–184, 1999.

[68] Chamila Wijayarathna and Nalin A. G. Arachchilage. 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*, EASE'18, page 205–210, New York, NY, USA, 2018. Association for Computing Machinery.

## A Task descriptions

These are the task descriptions as they were presented to participants:

**Create a method to hash and salt passwords for storage in a database.**

You are asked to develop a method in a web-application backend that prepares a password for storage in a database. Assume that a user has chosen a password that gets handed to your function hash_password() as a string. Implement this function in such a way that it returns the password securely hashed and salted. Additionally, please implement a function verify(), which compares a password to a hash and returns True if they match, False if they do not.

The programming language for this task is Python3. Please only use the website LetsHash as a resource when solving this task.

**When is the problem solved?**
The problem is solved when you have successfully implemented the function to fulfill the required specifications, and the output printed by the main function reads:
"Your hash: <a hash value>
The correct password is s3cr3t: True
The correct password is s3cr4t: False"

**Create a method to check if a password adheres to a given policy.**

You are asked to develop a method in a web-application frontend that ensures that the password a user chooses meets the company policy. According to this policy, a password must
- be between 8 and 64 characters
- have at least one upper- and one lowercase letter
- have at least one special character - have at least one number
Please implement the functions length() and composition() so they return True if the password matches the criteria given, and False if it does not.

The programming language for this task is JavaScript. Please only use the website LetsHash as a resource when solving this task.

**When is the problem solved?**
The problem is solved when you have successfully implemented the functions to fulfill the required specifications, and upon clicking "Run and Test", you receive an alert that reads:
"The password meets the requirement for length: true
The password meets the requirement for composition: false
The password is considered valid: false"

**Create a method to set up a second factor for user authentication.**

You are asked to develop a method in a web-application backend that offers users of a login system a second factor for authentication. Please implement the function generate_second_factor() so that it takes a shared secret as a parameter and returns a time-based one-time password (totp).

The programming language for this task is Python3. Please only use the website LetsHash as a resource when solving this task.

**When is the problem solved?**
The problem is solved when you have successfully implemented the function to fulfill the required specifications, and the output printed by the program reads:
"Your code: <a time-based one-time code>
Your code is verified: True"

## B Surveys

Some questions of the surveys were specific to either the groups LH and LH-W, or group C. These questions are marked accordingly.

**Survey**

Thank you very much for working on the tasks assigned to you during this study! There are a few questions we would like to ask you to wrap things up.

- (Q1): Please answer the following questions by indicating a number on the scale from "1 - Not at all familiar" to "7 - Very familiar".
  - How familiar are you with Python? *1-7*
  - How familiar are you with Javascript? *1-7*
  - How familiar are you with password storage in a database? *1-7*
  - How familiar are you with the implementation of two factor authentication (2fa)? *1-7*
  - How familiar are you with the implementation of password policies? *1-7*

- (Q2): Have you ever looked up how to implement password policies as they are recommended by any of the following institutions before this study? You can choose more than one answer.
  - No, I have never looked up recommendations on password policies.
  - I have looked up NIST‘s recommendations on password policies.
  - I have looked up OWASP‘s recommendations on password policies.
  - I have looked up another institution‘s recommendations on password policies. Please specify: *(Free text)*
- (Q3): Please rate the correctness of the following statements by indicating a number on the scale from "1 - Does not describe me" to "7 - Describes me very well".
  - I am familiar with the implementation of login forms. *1-7*
  - I am familiar with the implementation of password strength checkers. *1-7*
  - I have a good understanding of security concepts. *1-7*
- (Q4): How long were you actively working on the task to solve it? Please indicate the time in full hours. *(Free text)*
- (Q5): Please answer the following question by indicating a number on the scale from "1 - Very easy" to "7 - Very hard". Overall, the task was... *1-7*
- (Q6): Please rate the correctness of the following statement by indicating a number on the scale from "1 - Not close at all" to "7 - Very close". How close was the task to reality compared to the projects that you develop in everyday life? *1-7*
- (Q7): Did you have any prior experience with storing passwords in a database? You can choose more than one answer.
  - No.
  - Yes, in university.
  - Yes, on a job.
  - Other - please specify: *(Free text)*
- **Only groups LH/ LH-W** (Q8): If yes: Please rate the correctness of the following statement by indicating a number on the scale from "1 - Not at all helpful" to "7 - Very helpful". Would the website you have used in this study have been helpful in solving problems you had then? *1-7*
- (Q9): Did you have any prior experience with implementing password policies? You can choose more than one answer.
  - No.
  - Yes, in university.
  - Yes, on a job.
  - Other - please specify: *(Free text)*
- **Only groups LH/ LH-W** (Q10): If yes: Please rate the correctness of the following statement by indicating a number on the scale from "1 - Not at all helpful" to "7 - Very helpful". Would the website you have used in this study have been helpful in solving problems you had then? *1-7*
- (Q11): Did you have any prior experience with implementing two-factor authentication? You can choose more than one answer.
  - No.
  - Yes, in university.
  - Yes, on a job.
  - Other - please specify: *(Free text)*
- **Only groups LH/ LH-W** (Q12): If yes: Please rate the correctness of the following statement by indicating a number on the scale from "1 - Not at all helpful" to "7 - Very helpful". Would the website you have used in this study have been helpful in solving problems you had then? *1-7*
- **Only groups LH/ LH-W** (Q13): What could be improved about the website? *(Free text)*
- (Q14): Please answer the following questions by indicating a number on the scale from "1 - Never" to "7 - Every time".
  - How often do you ask for help when faced with security problems? *1-7*
  - How often are you asked for help when others are faced with security problems? *1-7*
  - How often do you need to add security to the software you develop in general (apart from this study)? *1-7*
- (Q15): Please answer the following questions by indicating a number on the scale from "1 - Not knowledgeable at all" to "7 - Very knowledgeable".
  - How would you rate your background/knowledge with regard to secure password storage in a database? *1-7*

  - How would you rate your background/knowledge with regard to the implementation of two factor authentication (2fa)? *1-7*
  - How would you rate your background/knowledge with regard to the implementation of password policies? *1-7*
- (Q16): How often have you stored passwords in a database in the software you have developed (apart from this study)? *(Free text)*
- (Q17): How often have you implemented two factor authentication (apart from this study)? *(Free text)*
- (Q18): How often have you implemented a login form with a password strength checker (apart from this study)? *(Free text)*
- (Q19): What is your most-used resource for implementing security in your software development?
  - Stackoverflow
  - Official documentation
  - Other - please specify: *(Free text)*
- **Only groups LH/ LH-W**:
  - (Q20-LH): Please rate your agreement to the following questions on a scale from "1 - Strongly disagree" to "7 - Strongly agree".
    * I needed a lot of background knowledge to complete the task. *1-7*
    * The website provided well-structured information. *1-7*
    * The website provided all necessary information to solve the task. *1-7*
    * I spent a lot of time trying to navigate the website. *1-7*
    * The assistance provided by the website to ease navigation was sufficient. *1-7*
    * I would recommend this website to a colleague who needs assistance with the implementation of password storage. *1-7*
    * I would recommend this website to a colleague who needs assistance with the implementation of two factor authentication. *1-7*
    * I would recommend this website to a colleague with questions regarding the implementation of password policies. *1-7*
    * I would use this website if I had to work on a similar task in a professional setting/ working on tasks within my job. *1-7*
  - (Q21-LH): Have you used only the website that was provided to you by this study? If not, which additional resources did you use to solve the tasks?
    * I have only used the website that was provided to me
    * I have used other resources as well: *(Free text)*
  - (Q22-LH): Please answer the following question by indicating a number on the scale from "1 - Much better" to "7 - Much worse". Compared to your most used resource, how would you rate the ease of use of the website you worked with during this study when it comes to accomplishing your tasks **functionally**? *1-7*
  - (Q23-LH): Please explain your decision: *(Free text)*
  - (Q24-LH): Please answer the following question by indicating a number on the scale from "1 - Much better" to "7 - Much worse". Compared to your most used resource, how would you rate the ease of use of the website you worked with during this study when it comes to accomplishing your tasks **securely**? *1-7*
  - (Q25-LH): Please explain your decision: *(Free text)*
  - (Q26-LH): Please rate your agreement to the following statements about the website that was provided for you during this study on a scale from "1 - Strongly disagree" to "5 - Strongly agree".
    * I think that I would like to use this website frequently. *1-5*
    * I found the website unnecessarily complex. *1-5*
    * I thought the website was easy to use. *1-5*
    * I think that I would need the support of a technical person to be able to use this website. *1-5*
    * I found the various functions in this website were well integrated. *1-5*
    * I thought there was too much inconsistency in this website. *1-5*
    * I would imagine that most people would learn to use this website very quickly. *1-5*
    * I found the website very cumbersome to use. *1-5*
    * I felt very confident using the website. *1-5*
    * I needed to learn a lot of things before I could get going with this website. *1-5*
- **Only group C**:
  - (Q20-C): Which resources did you use to solve the tasks? Please be as specific as possible (for example, provide links to any websites you used). *(Free text)*
  - (Q21-C): Which of the resources you listed in the last question was your main resource? *(Free text)*

- (Q22-C): Please answer the following question by indicating a number on the scale from "1 - Very good" to "7 - Very bad". How would you rate the ease of use of the website(s) you worked with during this study when it comes to accomplishing your tasks **functionally**? *1-7*
- (Q23-C): Please explain your decision: *(Free text)*
- (Q24-C): Please answer the following question by indicating a number on the scale from "1 - Very good" to "7 - Very bad". How would you rate the ease of use of the website(s) you worked with during this study when it comes to accomplishing your tasks **securely**? *1-7*
- (Q25-C): Please explain your decision: *(Free text)*
- (Q26-C): Please rate your agreement to the following questions on a scale from "1 - Strongly disagree" to "7 - Strongly agree".
  - I needed a lot of background knowledge to complete the task. *1-7*
  - The website(s) I used provided well-structured information. *1-7*
  - The website(s) I used provided all necessary information to solve the task. *1-7*
  - I spent a lot of time trying to navigate the website(s) I used. *1-7*
  - I would use the same website(s) if I had to work on a similar task in a professional setting/ was working on tasks within my job. *1-7*
- (Q27-C): Please rate your agreement to the following statements about the website that was your main resource during this study on a scale from "1 - Strongly disagree" to "5 - Strongly agree".
  - I think that I would like to use this website frequently. *1-5*
  - I found the website unnecessarily complex. *1-5*
  - I thought the website was easy to use. *1-5*
  - I think that I would need the support of a technical person to be able to use this website. *1-5*
  - I found the various functions in this website were well integrated. *1-5*
  - I thought there was too much inconsistency in this website. *1-5*
  - I would imagine that most people would learn to use this website very quickly. *1-5*
  - I found the website very cumbersome to use. *1-5*
  - I felt very confident using the website. *1-5*
  - I needed to learn a lot of things before I could get going with this website. *1-5*

- (Q28): Please select your gender.
  - Male
  - Female
  - Prefer not to say
  - Other: *(Free text)*
- (Q29): Please state your age. *(Free text)*
- (Q30): What is your current main occupation?
  - Freelance developer
  - Industrial developer
  - Industrial researcher
  - Academic researcher
  - Undergraduate part-time student
  - Undergraduate full-time student
  - Graduate part-time student
  - Graduate full-time student
  - Other: *(Free text)*
- (Q31): What is your nationality? *(Free text)*
- (Q32): How did you gain your IT skills? *(Free text)*
- (Q33): What was your main source of learning about IT-security? *(Free text)*
- (Q34): Do you have a university degree? *(Yes/ No)*
- If yes:
  - (Q35): What was/is your subject? *(Free text)*
  - (Q36): Were/Are you taught about IT-security at university? *(Free text)*
  - (Q37): Were/Are you taught about IT-security in addition to your regular studies? *(Yes/ No)*
  - (Q38): If yes: Where were/are you taught about IT-security in addition to your regular studies? *(Free text)*
- (Q39): Are you working at a company? *(Yes/ No)*
- If yes:
  - (Q40): How old is your organization? Please specify in years. *(Free text)*
  - (Q41): What is the total number of employees in your organization?
    - 1-9
    - 10-249
    - 250-499
    - 500-999

    - 1000 or more
  - (Q42): How many members are there in your team? *(Free text)*
  - (Q43): Which field of activity does your company belong to? You can choose more than one answer.
    - Game development
    - Development of network and communication software
    - Web development
    - Development of middleware, system components, libraries and frameworks
    - Development of other tools for developers, such as IDEs and compilers
    - Other: *(Free text)*
  - (Q44): Does your company have a security focus? *(Yes/ No)*
  - (Q45): Does your team have a security focus in its current field of activity?
    - Yes
    - No
    - I work alone and my field of activity has a security focus
    - I work alone and my field of activity has no security focus
  - (Q46): Do you also have to work on security-relevant tasks in your field of activity? *(Yes/ No)*
  - (Q47): Were/Are you taught about IT-security in addition to your regular work? *(Yes/ No)*
  - (Q48): If yes: Where were/are you taught about IT-security in addition to your regular work? *(Free text)*
- (Q49): What type(s) of software do you develop? You can choose more than one answer.
  - Web applications
  - Mobile/App applications
  - Desktop applications
  - Embedded Software Engineering
  - Enterprise applications
  - Other - please specify: *(Free text)*
- (Q50): How many years of experience do you have with software development in general? *(Free text)*
- (Q51): How many years of experience do you have with Python development? *(Free text)*
- (Q52): How many years of experience do you have with Javascript development? *(Free text)*
- (Q53): If you have any comments or suggestions, please leave them here: *(Free text)*

## C Security Score

For the security evaluation of the code, we used an adapted version of this security score from Naiakshina et al. [43]:

1. The end-user password is salted (+1) and hashed (+1).
2. The derived length of the hash is at least 160 bits long (+1).
3. The iteration count for key stretching is at least 1000 (+0.5) or 10000 (+1) for PBKDF2 and at least $2^{10}$ = 1024 for bcrypt (+1).
4. A memory-hard hashing function is used (+1).
5. The salt value is generated randomly (+1).
6. The salt is at least 32 bits in length (+1).

## D Automated Detection Of Copied Code

This section describes the process which was used to semi-automatically determine whether participants from the groups LH and LH-W submitted code which they had copied off of Let's Hash.

Since participants would sometimes copy everything, including comments, while others only copied the exact lines that were needed, exact string matching would have been too strict for our purposes. We used approximate string matching, coded in python, to calculate a matching ratio. If the resulting ratio dropped below a threshold of 80% for the cryptographic tasks, or 50% for the task on password policies, the files were examined manually. The thresholds were chosen on the basis of manual spot sampling. The threshold for the password policy task was much lower than for the other two tasks because this task involved some changes to the code that participants had to make to be able to use it, while the code for the other two tasks could be used as is.