“12345”, wherefore art thou accepted?

Measuring the current state of password requirements on websites

CS Graduation Assignment

by

Coen van Driel

Student number: 851698192
Course code: IM990C
Thesis committee: dr. ir. Hugo Jonker (chairman & examiner), Open University
dr. Fabian van den Broek (examiner), Open University

Open Universiteit
de beste www.ou.nl

Summary

Websites using a username-password authentication mechanism are vulnerable to password related vulnerabilities (e.g. by brute force attacks or password guessing attempts). To counter some of these vulnerabilities, websites can implement password input policies to enforce strong password creation by users. In some cases, websites are *required* to implement password policies (e.g. in regulated industries or by Europe's GDPR), In many other cases however, websites are not required to do so, shifting the responsibility of secure password creation to the visitor. Website designers can consult password policy standards and recommendations such as those from the NIST, OWASP, BIO and BSI for guidance.

We investigate to what extent websites adhere to these standards in order to protect their visitors. As website visitors are generally in control of the password they create, we focus specifically on websites that handle password related *input*. This consists of *composition* requirements and other input related requirements such as whether password-pasting is allowed, a common practice when using a password manager. We investigate both public websites aimed at the general public and private websites for internal use within organisations. Existing research in regards to analysing public websites is often performed with manual website analysis, a time consuming activity. Recent research is automating this type of analysis, but at the cost of automatically creating a large number of accounts on websites. We consider this an undesirable practice from an ethical perspective.

We therefore present a low-intrusive, client-side analysis based approach that locates registration pages, extracts client-side indicators (HTML attributes, textual instructions, and JavaScript code), and map these to standardized password rules. By way of a proof-of-concept we have demonstrated the feasibility of this approach at scale. We inferred password requirements without creating accounts on 8,254 websites, enabling large-scale and repeatable assessments.

We found that our approach provides the most value when measuring some individual requirements (e.g. min. and max. length) versus against complete standards that also include difficult to infer requirements. As a key finding, we found that 52% of websites that allowed for minimum length inference adhere to the NIST minimum length requirement, versus 48% non-compliant.

Aside from public websites, we also investigated private-organisational websites by way of two case studies. When comparing our findings to the current standards, we found that 100% of municipalities we surveyed and 85% of (applied sc.) universities are *not* compliant with the NIST standard, but *are* compliant to the local Dutch BIO 1.0 standard.

Contents

Chapter 1

Introduction

With currently more than 5 billion internet users,[1] the need for safe and secure interaction with websites and web-based applications (e.g. mobile apps) is more relevant than ever. An important part of many websites is the separation between authorized versus unauthorized data and functionality. This creates the need for user-authentication mechanisms. There are various different authentication mechanisms a website can offer its visitors. The username- password mechanism for authentication is a commonly used solution, that based on a 2024 report,[2] is the de-facto standard.

The username-password mechanism has its share of vulnerabilities. Its primary vulnerability is 'weak' password usage.[3] Weak passwords are easy to guess by attackers, leaving users at risk. Websites and password-using applications can counter this vulnerability by implementing password input requirements to avoid weak password creation. Password input requirements largely consist of composition requirements such as minimum length. Allowing a visitor to paste a password from, for example, a password manager can also be considered an input requirement. Potential additions to the username-password mechanism such as 2-factor authentication are still lacking as a 2024 report demonstrates.[4] Successors to passwords such as *passkeys*[5] are increasing in popularity, but are still far from being universally adopted with only 12% of the top 250 websites supporting passkeys. [6]

This demonstrates the continued relevance of judiciously implemented password requirements for password security.

Password restrictions depend on targeted user

In this study, we investigate password input requirements on websites. As websites can target different audiences, they therefore may also differ in their password requirements depending on the targeted user. Websites used in an organisational context may have more stringent password requirements as a breach impacts the

[1] https://www.statista.com/statistics/617136/digital-population-worldwide/

[2] https://fidoalliance.org/wp-content/uploads/2024/10/Barometer-Report-2024-Oct-29.pdf

[3] https://www.forbes.com/advisor/business/software/american-password-habits

[4] https://www.gov.uk/government/statistics/cyber-security-breaches-survey-2024/cyber-security-breaches-survey-2024

[5] https://blog.google/technology/safety-security/one-step-closer-to-a-passwordless-future/

[6] https://fidoalliance.org/wp-content/uploads/2024/05/World-Password-Day-2024-Report-FIDO-Alliance.pdf

whole organisation. This as opposed to a website targeting individuals, where a breach may only impact a single user account.

We distinguish between **public websites** (generally, business-to-consumer sites) such as webshops, news websites, e-magazines, and social media and **private-organisational websites**: generally, B2B SaaS sites and employee-only sites such as intranets. As passwords are used broadly across either setting, we investigate tendencies in password input requirements in both settings. We study password restrictions in public websites by automated analysis, and in private-organisational sites via two case studies: sites by higher education institutes (regular universities and universities of applied science) by desk research, and sites by municipalities by surveying professionals. To determine what password requirements are of relevance to websites, we investigate the password requirement guidelines by multiple standardization bodies such as the NIST.[7]

Studying password restrictions imposed on public websites. For measuring password requirements on public websites, we aim to design a method for *automatically* inferring the password input requirements of a given website. Other studies attempting to automate this type of website analysis such as that by Alroomi and Li [AL23] has resulted in successful analysis at a larger scale, but at the 'cost' of creating many accounts on websites to simulate user sign-ups (i.e. an 'account creation approach'). We object to this approach as filling in registration forms automatically is not only error-prone, it can also inflict undue burden on the receiving website. We therefore aim to devise a more ethical analysis method, one that avoids undue burdens for the analysed websites.

Investigating private-organisational websites. Aside from public websites, password (input) policies are also relevant in a non-public context. Websites in a private-organisational context are different from public websites as their account creation process is generally shielded from public access. This results in limitations for discovering their password requirements and thus requires a different approach than automated scanning.

To the best of our knowledge, the research domain of password policies within a private, organisational, context has only been given scant attention. This despite these policies being of great importance to an organisation's risk profile as most organisational data breaches are password vulnerability related.[8,9]

We use two approaches, first, we analyse publicly available security policy documents in the Dutch educational sector. Second, we survey civil servants at Dutch municipalities responsible for their organisation's password policies. We provide additional context (e.g. the basis for these policies) and map the results to current password standards.

[7] `https://www.nist.gov/`

[8] https://www.verizon.com/business/en-gb/resources/2022-data-breach-investigations-report-dbir.pdf

[9] https://inquest.net/wp-content/uploads/2023-data-breach-investigations-report-dbir.pdf

Research questions. The above leads us to the following main research question:

> **To what extent are password input requirements of websites compliant with password standards?**

For public websites we aim at analysis at scale. This necessitates the automation of password requirement inference. Unlike public websites, insights into private-organisational websites' policies may be less accessible. We therefore investigate public and private-organisational websites separately.

Due to this, we divide the main question into the following sub-questions:

RQ 1. How to automatically infer password input requirements from public websites, at scale, in an ethical manner?

RQ 2. To what extent are public websites compliant to the password input requirements of password standards?

RQ 3. To what extent are private-organisational websites compliant to the password input requirements of password standards?

Contributions. This study contributes to the cybersecurity domain by presenting a new method for inferring password requirements from public websites. Additionally this study presents new insights into password requirement compliance of both public websites and password requirement compliance in a private, organisational context. This study contributes the following:

1. We present an automated, low-intrusive, method for inferring password requirements from public websites solely based on client-side source code. This works by capturing relevant client-side password requirement indicators, using an indicator mapping and interpretation rules to infer password requirements from the captured indicators.
2. We create a proof-of-concept implementation of our method and use this to scan 8,254 websites. We relate these findings to the following password requirement standards: BIO, BSI, NIST and OWASP. When looking at individual password requirements, the minimum length requirement is *most* adhered to by 52% of websites when compared to the NIST standard.
3. In addition, we investigate password policies used on internal systems. We do so by performing desk research and surveying security professionals responsible for managing these policies in the education sector and Dutch municipalities. We find that 100% of municipalities and 85% of (applied sc.) universities are not compliant with the NIST standard, but are with the local BIO 1.0 standard.

Collectively, these contributions provide a new, more ethical, approach to website analysis at scale and insights into password composition requirement compliance of websites in both a public and private context.

Chapter 2

Background

Password input requirements are a series of criteria that a password has to meet in order to be accepted by the website. Website developers and designers are free to use the requirements as they see fit. This ranges from composition requirements such as minimum length, to extended validation requirements such as blocking specific keywords in passwords. An example of a website demonstrating password requirements is LastPass, as shown in Figure 2.1.

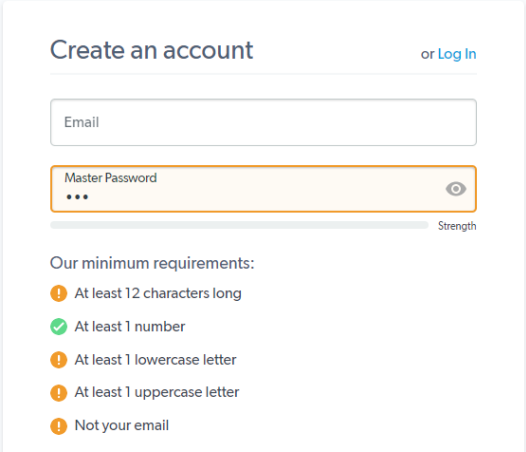


Figure 2.1: LastPass password requirements

LastPass in the example above is using both composition (length and character type requirement; a number, upper and lowercase letter) and a context-based requirement (the password must not match the user's email address). These requirements are intended to force website visitors to avoid common mistakes when creating passwords that can lead to weak passwords, and therefore become a vulnerability to the user's account.[1]

The *strength* of a password is a measure of how difficult a password is to 'guess' or 'crack' by attackers.[2] This 'guessing' can take the form of literal guessing by a person (e.g. trial and error, social engineering), or a more systematic approach (e.g. brute force password hacking using automated tooling).

[1] `https://cwe.mitre.org/data/definitions/521.html`
[2] `https://www.sciencedirect.com/topics/computer-science/password-strength`

Websites that have password requirements judiciously implemented, are at lower risk of a data breach due to weak passwords. Overcomplicating password requirements to minimize cybersecurity risk however is not the solution. Both ends of the spectrum (from no requirements to very complex requirements) are proven problematic.[3] The NIST[4] standard mentions overly complexity requirements as counterproductive for security as it can steer people into trying to circumvent these requirements and thus use less safe passwords [KSK+11]. The key to secure password requirements is therefore to balance complexity and usability.

2.0.1 Current password standards and guidelines

Website designers can look at leading institutions and authorities in the field of cybersecurity to determine which password requirements are effective (i.e. recommended). We consider an institution or authority in this field leading when it provides guidelines or rules for password requirements that are broadly recommended and adopted.

In this study we focus on the following institutions and authorities that provide password requirement standards or guidelines:

- **NIST**, National Institute of Standards and Technology;
- **OWASP**, Open Worldwide Application Security Project, in this study specifically the OWASP Application Security Verification Standard;
- **BIO**, 'Baseline Informatiebeveiliging Overheid', Dutch Government information security baseline;
- **BSI**, 'Bundesamt für Sicherheit in der Informationstechnik', German Federal Office for Information Security.

These standards were selected because they each represent authoritative and widely adopted standards or guidelines in different international and local contexts, both for public websites and organisations.

The input requirements of the NIST[5], OWASP[6], BIO[7] and BSI[8], are categorized and displayed in Table 2.1 starting with the most recent. We have listed the latest versions of each standard.

[3] https://www.doioig.gov/sites/default/files/2021-migration/Final%20Inspection%20Report_DOI%20Password_Public.pdf

[4] https://pages.nist.gov/800-63-3/sp800-63b.html

[5] https://pages.nist.gov/800-63-3/sp800-63b.html

[6] https://github.com/OWASP/ASVS/blob/master/4.0/en/0x11-V2-Authentication.md#v21-password-security-requirements

[7] https://cip-overheid.nl/media/uvplkjfz/20231106-bio-thema-uitwerking-toegangsbeveiliging-v2-3-def.pdf

[8] https://www.skadden.com/-/media/files/publications/2024/04/data-protection-rulings/the-german-federal-office-for-information-security.pdf

Requirement	BIO 1.0 (NL,2023)	BSI (DE,2022)	NIST (US,2020)	OWASP (US,2019)
Composition				
Minimum length	8	8	8	12
Maximum length	–	–	≥ 64	≥ 64
Require character-types	Y	Y	N	N
Allow spaces	–	Y	Y	Y
Reject sequential characters	o	o	o	Y
Miscellaneous				
Allow password pasting	–	Y	Y	Y
Reject known bad passwords	–	Y	Y	Y

Y: required; N: forbidden; o: optional; –: not specified

Table 2.1: Password input requirements in various standards

The password requirements in Table 2.1 have been categorized in composition and miscellaneous. In the composition category, *require character-types* refers to a password complexity requirement, where some or all, character-types such as lowercase letters, uppercase letters, numbers and special characters are required. This requirement is aimed at enforcing password *complexity* (i.e. rejecting simple dictionary words). The requirement *reject sequential characters*, refers to the blocking of characters that logically follow up after each other (e.g. 12345, abcde, qwerty).

A note on BIO's and BSI's minimum length requirement: a distinction is made between regular user accounts and privileged accounts. Where as for regular accounts the minimum length is 8, for privileged accounts it is increased to 12. In this study we do not specifically investigate high risk environments such as admin panels for which privileged accounts are needed, but more employer to employee-websites as a whole. Therefore we include the minium length of 8. On a similar note, BIO also allows the complexity (i.e. character-types) requirement to be ignored when minimum length is configured to at least 20 characters).

In regards to the miscellaneous requirements in Table 2.1:

- *Allow password pasting*, referring to the use of pasting copied passwords, for example when using password managers.
- *Reject known bad passwords*, referring to the blocking of words commonly used (e.g. dictionary words), passwords to are known to be common (e.g. password123, mypassword) and passwords that have been uncovered from successful hacks.

More stringent password requirements are in the works. The most recent password standard in Table 2.1 is from 2023. Newer versions of these standards and guidelines are in the works as version 2.0[9] of the BIO is expected to be published in late 2025. In the BIO 2.0 version the minimum length increases to 15, and the password complexity requirement is dropped. For NIST a similar update is expected

[9] `https://bio-overheid.nl/bio-wijzigingen`

as a 2025 blogpost[10] from NIST recommends a minimum length of 15 characters. In this study we only use the current, officially published versions of the standards and guidelines.

Non-input requirements. As mentioned, websites are free to create their own password requirements. These 'custom' requirements may be based on previous breaches or specific threats to their case, or in fact, just because the website designers think it's a good requirement. An example of this can be found on the University of Ljubljana's website as shown in Figure 2.2. This example prohibits the use of the keywords "*script, select, insert, update, delete, drop, –, ', /*, */*", referring to commands used when exploiting SQL injection vulnerabilities.

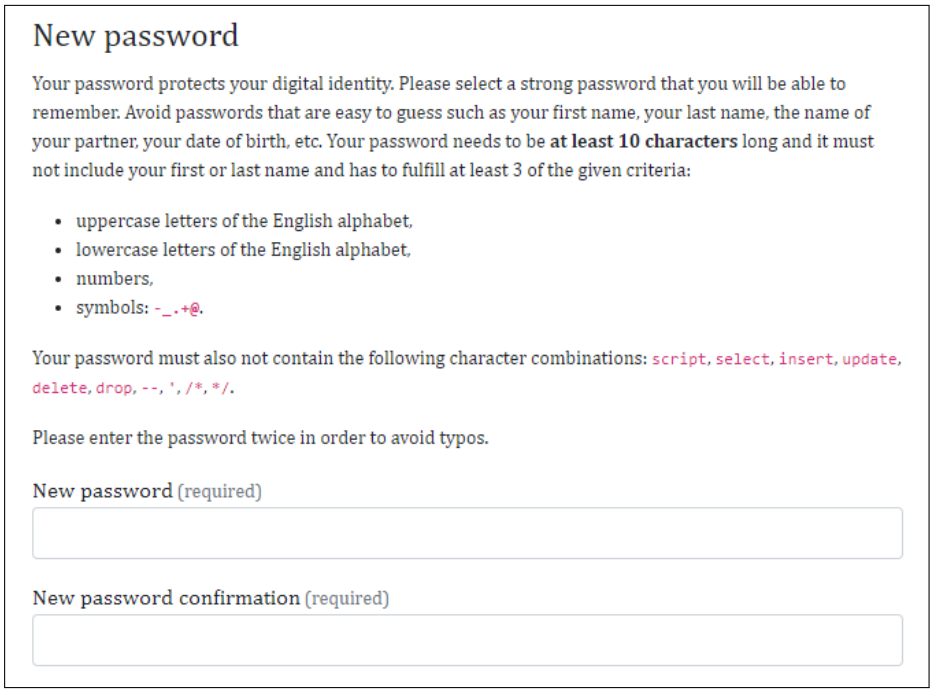


Figure 2.2: University of Ljubljana password requirements, as discovered by Hugo Jonker

[10] `https://www.nist.gov/cybersecurity/how-do-i-create-good-password`

2.1 Password requirements implementation techniques

When automating the determination of password requirements of websites, a technical understanding of the implementation of password requirements is foundational. When looking at the password mechanisms for websites, we can distinguish two separate layers within the structure of a website; **client-side** code and **server-side** code. Both layers are suited for the implementation of password policies, even though client-side scripts have grown in popularity due to its improved usability due to immediate feedback capabilities to website visitors.

These layers can, independently or combined, provide password requirement enforcement (i.e validation) logic, but also assist visitors with instructions and visual cues in regards to the policies in place (e.g. display a list of requirements).

2.1.1 Key distinguishments

Prior to further elaborating on password requirement implementation techniques, it is important to distinguish and define several components that are relevant. In this study we solely focus on the client-side layer of a website, as opposed to the server-side. We define these terms as follows:

Client-side and server-side

Client-side. Also known as front-end, refers to code that is running in the website's visitors web-browser, this code (or script) can execute actions without performing network requests to the website's server. This code is generally downloaded to a website's visitor's web-browser when opening the website or is downloaded on certain user actions. The primary and effectively only client-side programming language is JavaScript. However, newer versions of `HTML` and CSS also support some basic logical operations (e.g. conditional CSS, `HTML` RegEx validation). Besides JavaScript there are some new client-side programming language initiatives such as WebAssemly,[11] but adoption is slow.[12]

Server-side. Also referred to as back-end, is code that executes as a response to a network request to the website's server. The actual code that is executed is hidden from the website's visitor and the web-browser itself, only the finalized response after the server-side code is executed is returned to the web-browser. Examples of server-side programming languages are PHP, .NET and C++. A common practice is to combine server-side with client-side validations, as website visitors *can* bypass solely client-side validations. When combining both server-side and client-side validations, discrepancies can occur. These possibly discrepancies are something to further analyse when validating our analysis method.

[11] `https://developer.mozilla.org/en-US/docs/WebAssembly`
[12] `https://thenewstack.io/webassembly-adoption-is-slow-and-steady-winning-the-race/`

Instructed and enforced password requirements

A second key distinguishment is the difference between *enforced* versus *instructed* password requirements.

Instructed password requirements. Instructed password requirements are often visually displayed to the user during the account/password creation phase. These instructions aim to provide the user with visual hints or instructions to what passes as an accepted password. The actual enforcement may differ from these instructions due to design/programming discrepancies.

Enforced password requirements. Enforced password requirements are executed as a separate validation that validates the user input before it passes further into the system. This validation can be aimed at ensuring user input adheres to functional requirements such as as password policy, but also to protect the underlying system (e.g. the website's user database) for malformed input and prevent errors.

Enforced password requirements are more valuable than instructed password requirements when determining requirements, as these are what actually is accepted by the website.

2.1.2 Client-side password requirement indicators

As we focus on client-side solely in this study, we are interested in website elements that *can* exhibit information regarding both instructed or enforced password requirements. For this context, we refer to all client-side elements that contain information regarding password policy requirements as *'password requirement indicators'*.

We distinguish the following password requirement indicators:

- **HTML attributes:** `minlength`,[13] `maxlength`,[14] `pattern`,[15] **and** `passwordrules`.[16] The latter is in the proposal phase, the others are accepted standards indicating minimum length, maximum length, and a regular expression to which the input must conform, respectively.
- **Textual password instructions**
  Interpreting textual instructions may also provide information regarding a websites server-side validations. The drawback of textual messages is that there is no certainty that the client or server-side validation is aligned with these messages.
- **Client-side dynamic feedback**
  Password requirements can be hinted at or enforced by client-side scripts. Examples include showing a password strength meter, or showing requirements that are not yet fulfilled in red, and the satisfied ones in green. For our purpose, extracting the underlying logic from client-side dynamic feedback would require not only parsing the script, but also extrapolating the intentions that it encodes. We distinguish two types of scripting, RegEx validation (e.g. input is checked against a RegEx-pattern), and algorithm based scripting (e.g. a JavaScript method that checks the length of a password).

In order to automatically determine a website's password requirements, all three password requirement indicators can be analysed individually or in combination. An analysis of all three indicators will yield the most complete result as not all websites by definition will have implemented all, or any of these client-side elements.

Conflicting indicators. A website can use multiple password requirement mechanisms, e.g. textual instructions *and* `HTML` attributes. In some cases, these mechanisms may conflict with each other. For example, a textual instruction for a minimum length of 12 conflicts with `HTML` attribute `minlength='8'`. An example of such a possible conflict is shown in Figure 2.3, in this case a validation is conflicting with the password instructions. For automatic inference of password requirements, such conflicts must be automatically (1) detected, and (2) resolved. These conflicts can be resolved in various ways. For example, an enforced indicator (e.g. sourced from JavaScript) can take precedent over an instructed indicator (e.g. sourced from textual instructions). A 'rulebook' to automatically determine in what way conflicts are resolved is needed.

[13] `https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/minlength`
[14] `https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/maxlength`
[15] `https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/pattern`
[16] `https://github.com/whatwg/html/issues/3518`

Figure 2.3: Red textbox indicates bad password, while all requirements are met (seen on odido.nl, 2023-10-25)

Chapter 3

Related work

When we specifically look at determining password requirements of websites and password related vulnerabilities through website analysis, a number of studies have been performed.

Manual analysis studies. The most recent and most notable study in terms of manual analyses was performed by Lee et al. [LSN22] Lee et al. have completed this analysis by manually analyzing 120 of the most popular websites and comparing the password requirements against the password composition best practices, in this case the NIST standard. Lee et al. found that all 120 analysed websites performed poorly. Lee et al. found that more than half of the websites studied do not have strict password policies and allow commonly breached and commonly used passwords (e.g., '12345678', 'rockyou').

This study has analysed 120 websites, in a manual, labour intensive way of analysing, which is a significant drawback. Lee et al. indicated that this manual research method was the only method to ensure high accuracy of their findings as they found no uniform indication in which websites declare their password requirements.

Other studies [MKV17, SHPS17, WW15] came to similar findings by way of manual analysis, but these are more dated and may not represent the current state of password requirements anymore. Alroomi and Li [AL23] investigated more studies in this domain and found that no study, including studies stemming from 2007 and newer, exceeded the analysis of more than 150 websites.

Automated analysis studies. As established prior, research in this specific domain is not new, but has always been limited due to automation difficulties. Even the recent study by Lee et al. [LSN22] attempted automation, but experienced many difficulties and proceeded with manual website analysis instead. Studies that *did* focus on large-scale automated website analysis did so by mostly focusing on scanning for vulnerabilities of the *login-page*.

For example, Van Acker et al. [VHS17] performed a large-scale study on the Alexa top 100,000 websites. Van Acker et al. scanned for login-page vulnerabilities such as un-encrypted login-form submitting. A key element of this study is the methodology for finding the login-page, which is of relevance to our efforts in this study.

Jonker et al., [JKKS20] with the introduction of the Shepherd framework, successfully demonstrated a methodology for automated website analysis in order to

scan a larger set of websites. Similar to Van Acker et al., the Shepherd project demonstrated ways to find login forms to then analyse its security properties.

Calzavara et al. [CJKR21] further built upon the Shepherd framework to analyse session management vulnerabilities of websites as part of the login, logout and authentication cookie handling process. These studies focused on designing methodologies for discovering vulnerabilities in login pages.

Other studies specifically focusing on the automation of the website registration process are limited. Alroomi and Li [AL23] was, to the best of our knowledge, the first study to successfully apply browser automation at scale for this specific type of analysis. This study's methodology is in essence a literal automation of the prior studies that were performed manually. This approach to automation (i.e. simulating a website user's behaviour through automation) provides benefits such as high accuracy, but also drawbacks such as the 'cost' of account creating at scale.

Studies in the organisational context. In a German study by Gerlitz et al. [GHS21] password policies in organisations were analysed and found that a number of organisations still enforce harmful policies such as *password expiration*, which newer standards such as the NIST 2020, standard and commercial baselines such as Microsoft's baseline[1] are moving away from. This is partly due to a German information security standard (BSI[2]) that contains this recommendation while other standards have dismissed this recommendation.

Organisations can benefit from standardizing password policies as demonstrated by Mikko et al.[Y+04]. Technologies such as Microsoft's Active Directory and newer cloud based identity-services can create a single source of identity- and password-management to use as Single Sign-On which increases control over passwords and their required strength (e.g. single source of password requirements for all applications, single reset point, single policy point). For other websites and applications where Single Sign-On is not available, unique strong passwords and the use of password managers is the recommended solution, similar to that of public websites that do not offer (Social) Single Sign-On.

[1] `https://learn.microsoft.com/en-us/archive/blogs/secguide/security-baseline-final-for-windows-10-v1903-and-windows-server-v1903`

[2] `https://www.bsi.bund.de/EN/Home/home_node.html`

Part I

Websites targeting the general public

Chapter 4

Methodology

In order to answer the main research question and its sub-questions, a research strategy must be devised. For ease of readability we refer back to the research questions as introduced in Chapter 1.

Main research question

> **To what extent are password input requirements of websites compliant with password standards?**

In this part of this study, we investigate public websites aimed at the general public. This requires analysis at scale and therefore a technical approach. In this part we investigate sub-questions **RQ 1** and **RQ 2** from the sub-questions below:

RQ 1. How to automatically infer password input requirements from public websites, at scale, in an ethical manner?

RQ 2. To what extent are public websites compliant to the password input requirements of password standards?

RQ 3. To what extent are private-organisational websites compliant to the password input requirements of password standards?

Ethical considerations. We are interested in analysing websites in an ethical manner and avoiding ethical 'gray areas' (e.g. using data from illegal sources). In related studies, whether though manual [LSN22] or automated analysis [AL23], accounts can be created on websites to determine password acceptance bounds and thus password requirements. 'Account creation'-studies may provide highly accurate results as it simulates user behaviour, but at the cost of creating, a possible high number of, unused accounts.

The created accounts *can* be used by websites to gage growth and the need for additional resources (e.g. compute and storage). For example, a website offering storage services maybe reserve a base storage capacity for each created user. 'Account creation'-studies therefore may unjustly impact resource reservation-planning, resulting in waste and cost. This method of analysis tends to also violates a website's Terms of Service – it is even considered a misuse vector by the *OWASP Automated Threat Handbook (Account Creation:OAT-019)*.[1]

To this end, our selection of methodology must be aligned with a more ethical method of website analysis.

[1] `https://owasp.org/www-project-automated-threats-to-web-applications/`

Selection of methodology and alternatives. To infer the password requirements of public websites, multiple strategies are available. The strategies we considered and selected are listed below:

- **Analysing passwords from password sharing websites**
  Resources like *BugMeNot*[2] publicly list usernames and passwords of websites for account sharing purposes. These publicly available accounts can be analysed for password structures and be used for reverse engineering password requirements from known accepted passwords.
  Resources such as BugMeNot however, have limitations as websites can opt-out, and account sharing is often not allowed by websites, placing the use of this type of resource in a legal gray area. Perhaps more importantly, these 'accepted' passwords may actually be invalid or no longer meeting newer password requirements. This invalidity was demonstrated by Calzavara et al. [CJKR21]. Their analysis showed that 66% of the passwords from BugMeNot were no longer accepted during login-attempts. We therefore consider this approach not suitable.
- **Analysing existing password from hacked password lists**
  Similar to the previous *BugMeNot* approach, password lists that have been publicized after hacks can also be used to reverse engineer password requirements. This approach however has two drawbacks. First, password lists are generally based on specific websites. In our study we are interested in a broad approach to inferring password requirements. Hence we are are only interested in an approach that is based on many different websites, as opposed to a small number of sites that has their passwords exposed. In general we consider using password lists resulting from (illegal) hacks an unethical practice as this also moves into the illegal territory. Using data from hacks indirectly legitimatizes the original hack or breach and contributes to the idea that all data is 'free to use' regardless of its origin. We therefore consider this approach not suitable.
- **Making use of the analysis of website components**
  Tooling exists that analyses websites and determines its technical components, such as *'BuildWith Technology Lookup'*.[3] Based on these found components, we can aim to infer the default password requirements a component uses (e.g. a specific JavaScript framework).
  A closely related alternative method is to analyse commonly used frameworks such as CMS (Content Management Systems, e.g. WordPress) or even specific open source libraries (e.g. strength-meter libraries) and analyse its usage numbers(for example, how many websites on the internet use Wordpress?).
  Website data collected from this approach can then be matched with the current password standards to answer **RQ 2**. Both these approaches benefit from existing data, but focuses on high level website components. This assumes that the *default* password requirements are what is actually implemented. This may lead to inaccuracy and questions the validity of the results.
- **Designing a novel, more ethical, approach for inferring password requirements**
  In this study we aim to automate password requirement using a more ethical

[2] `https://bugmenot.com`
[3] `https://builtwith.com`

method (i.e. website analysis while avoiding account creation). To the best of our knowledge this has not been performed yet.
We aim to design a method that identifies and interprets password input requirement *indicators* using only a website's client-side source code. Client-side password requirement indicators are code or text elements in HTML or JavaScript that contain information regarding password requirements. The downside is that a client-side-only approach increases the risk that our analysis is incomplete or even faulty. We can address this by first manually analysing websites to establish a ground-truth.
By way of proof-of-concept we can validate our approach and collect password input requirements from a large number of public websites across the web. With these findings we can measure compliance to the current password standards to answer **RQ 2**. This approach is aligned with our ethical considerations and research questions, *we therefore consider this approach suitable and choose this approach moving forward.*

Our selected, client-side based analysis approach, creates significantly less burden on websites than the 'account creation' approach. Some burden however is unavoidable, such as using website bandwidth from opening and downloading resources. We consider this acceptable in relation to the value this study offers. Our approach is also significantly less susceptible to malicious use by bad actors as it does not automate credential handling (i.e. cannot be used for brute force attacks) and does not create accounts (i.e. cannot be used for resource exhaustion attacks).

Additionally, during the planning phase of this study, we requested the Open Universities board of ethics (ERBi) for feedback (found in Appendix A). In our review request, where we considered both an 'account creation' and 'non-account creation' (i.e. client-side based) approach, we used the Menlo Report[4] as a framework for determining ethical aspects.

For our client-side based approach, we consider there to be no significant ethical objections to present our approach and its proof-of-concept publicly.

[4] `https://www.dhs.gov/sites/default/files/publications/CSD-MenloPrinciplesCORE-20120803_1.pdf`

Chapter 5

Determining password indicator usage on websites

Determining what indicators are used on websites and how frequent can be considered our 'ground truth' and help shape the design of our methodology. In order to determine which password indicators are used, we perform manual website analysis.

5.1 Approach to website analysis

In Section 2.1.2 we listed three client-side password indicators websites can use (`HTML`-attributes, textual instructions and hints, client-side scripts for dynamic feedback). In our website analysis we set out to discover which of these client-side password indicators are used. We aim to discover usage patterns (i.e. the frequency of specific indicator usage), in order to provide guidance for designing the methodology (e.g. which indicator will yield the most results).

Website selection & exclusion. To gather the required information, we analyse a number of popular websites and search their client-side source code for password requirement indicators.

Because this method is performed manually, and limited by time of the study, the execution is limited to a 100 websites. These 100 popular websites (that offer free account registration) include a wide range of different website types. Websites included in the top 100 for example are social media (e.g. Google.com and Facebook.com) websites, as well as news outlets and entertainment websites (e.g. Dailymail.co.uk) and business to business or commercial services websites(e.g. Hubspot.com). Our top 100 website list is based on the most popular websites of the Tranco-List.[1] Alternatives for the Tranco-list exist, such as the OpenRank-list.[2] The Tranco-list however is the current standard for research use as this list is hardened against manipulation. Because this research is performed by the research team themselves only English and Dutch websites are analysed. For a website from the Tranco-list to be eligible for analysis, it must meet the following criteria:

- The website is considered *safe*, that is:
  - The site is not a known spreader of malware or phishing,

[1] `https://tranco-list.eu/`

[2] `https://www.domcop.com/openpagerank/what-is-openpagerank`

  - The site is not an adult-entertainment site.

  see below for details.
- Language of the website is English.
- The website has a publicly accessible account registration option.
- Registration only requires an email address and data that can be mocked/unverified (e.g. data of birth).
- Registration is free or trial (i.e. requires no direct payment).

Browser safety configuration & website safety filtering. To ensure that the website analysis does not harm the computers of the research team the analysis is performed on AVG secure browser[3], a safety-enhanced version of Chrome, is used with *Web Shield* enabled to visit the selected websites.

AVG secure browser is almost identical to Google Chrome but with more security enhancements. Both browsers are based on the same Chromium[4] foundation.

Additionally, for a website to be eligible for analysis all sources below must provide a 'Safe'-rating and not be categorized as Adult-websites as these are high risk websites from both a malware perspective as a legal perspective.

- https://www.virustotal.com/gui/home/url
- https://sitecheck.sucuri.net
- https://global.sitesafety.trendmicro.com

Performing the website analysis. Website analysis is performed by applying the following steps:

1. **Locate account registration page**
   The initial step on the selected website is to locate the registration page. In general, we seek an option to create an account by registering using an email address (as opposed to Single Sign On methods).
2. **Using Browser DevTools to analyse `HTML`**
   The `HTML` source code is the first client-side source code to analyse, and we start by locating the password-field element. We can easily locate the password-field element using DevTools by selecting the password-field in the user interface and inspecting it through DevTools. We check the `HTML` element for HTML-attributes such as `minlength` and `maxlength` and search for other attributes that may indicate password requirements. We then search the `HTML` source code using the DevTools text-search feature and searching for password related keywords (e.g. 'password', 'characters', 'length', 'symbols'). This tactic can point us into the direction where password related `HTML` elements are located (e.g. password instructions). By taking an exploratory approach, we also aim to discover how we can programmatically identify these elements, with the automation phase of our study in mind.
3. **Using Browser DevTools to analyse JavaScript**
   Additionally to analysing `HTML` client-side source code, we also manually analyse any existing JavaScript client-side code. JavaScript on a website can be within the same `HTML` page as in-page, or in-line JavaScript and is analysed

[3] `https://www.avg.com/en-us/secure-browser`
[4] `https://www.chromium.org/chromium-projects/`

similarly as HTML. JavaScript can also be attached as separate files, which can also be accessed through DevTools. Within these files we search for password related keywords, similar to our HTML text-search approach. Here we also take an exploratory approach to determine both indicator usage, as well the discovery of patterns for identifying indicator-elements.

Our aim is to limit website interaction as this might influence client-side indicators. Our analysis does *not* include the creation of accounts on the websites under test. We have noticed however that websites sometimes do not load password requirement indicators into the client-side source code until the password field is focused on (e.g. clicked on). Therefore, during our client-side analysis, we place browser focus on the password field to optimize our collection of available client-side indicators. As no accounts are created during our analysis, we consider our analysis acceptable from an ethical perspective.

5.2 Results

5.2.1 Estimating incidence of client-side elements

We performed our manual website analysis and tracked our results when we found password requirement indicators. The complete results of our analysis of 100 websites can be found in Appendix B; in Table 5.1 we summarise the results. Websites can exhibit multiple indicators, hence overlap between indicators is possible. For textual password instructions we distinguish between text found within HTML tags versus text found within JavaScript tags, a single website may contain both.

Found indicator	#
Textual password instructions	63
– within HTML elements	44
– within JavaScript tags	21
HTML attribute `maxlength`	26
HTML attribute `minlength`	3
HTML attribute `pattern`	3
HTML attribute `passwordrules`	1
Dynamic feedback logic	3
– JavaScript validation algorithm	2
– JavaScript validation RegEx	1

Table 5.1: Number of client-side indicators found on a set of 100 websites

5.2.2 Discussion

Noticeable is the high number of websites displaying textual password instructions versus other indicators. What is also noticeable is that the other types of password requirement indicators were not found with any significant consistency.

`HTML maxlength` being the most frequently found requirement indicator next to textual instructions, is surprising from a password requirement perspective. In terms of password strength, having a sufficient `minlength` requirement adds more strength than maximum length requirement.

The number of websites that have textual password instructions in either `HTML` or JavaScript, is sufficiently high in order to proceed with our methodology of inferring password requirements. These results can be used to build indicator repositories which then be can mapped to password requirement specific mapping rules in order to infer a website's password requirements.

Our results are limited by several aspects however. We noticed that websites using Single Page Application-frameworks such as React or Angular were difficult to manually 'read' in order to find requirement indicators. Additionally, websites using external JavaScript libraries, especially in the form of many modular components on Content Delivery Networks, were also more difficult to determine its relevance and process flow. These difficulties may have resulted in not discovering all client-side password requirement indicators.

Chapter 6

Automatically determining password client-side indicators

In this chapter we present the first step towards automation. We present the *first step* of our automated password requirement inferring-approach; namely an automated approach for determining which indicators are used on a website.

This step allows us to automatically build client-side password indicator *repositories* (i.e. a data store for indicators), from which we can infer password requirements in the next step. The approach presented in this chapter only refers to automatically finding indicators on websites. The inferring of password requirements will be presented in Chapter 7.

In Figure 6.1, we present an overview of our approach to determine the usage of password client-side indicators.

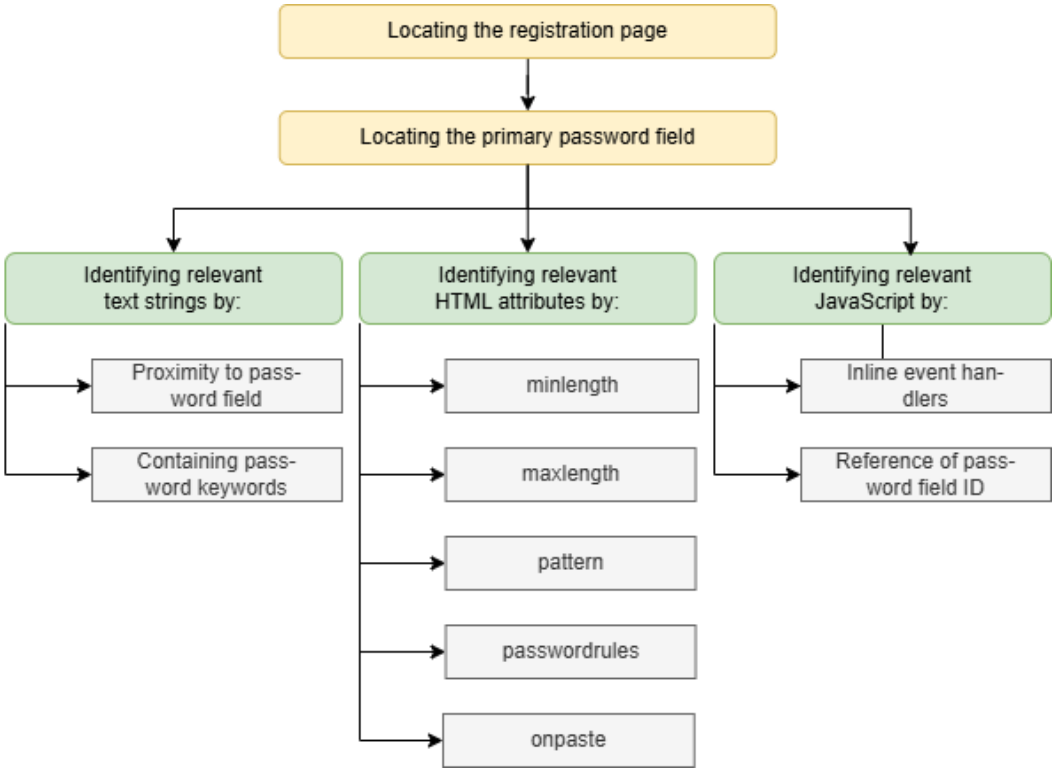


Figure 6.1: Automatically determining client-side password indicator usage

6.1 Locating the registration page

In this study we are solely interested in the registration page of a given website. To locate and navigate to this page, we perform a search process similar to that of Jonker et al. [JKKS20] and Alroomi and Li. [AL23]. This search process is performed by way of the following steps listed below. A subsequent step is only performed when the previous step is not successful in locating the registration page.

1. **URLs with registration keywords**
   We try to find and open registration pages based on common URL patterns such as {base-url}/register and {base-url}/sign-up.
2. **Landing page search**
   On the landing page we search for keywords and accompanying hyperlinks related to registration (e.g. 'register', 'sign up', 'create').
3. **Google Search**
   We use Google Search using webbrowser automation and search for the website base-URL in combination with keywords related to registration (e.g. 'register', 'sign up', 'create').

When a possible registration page is detected, we automatically navigate to this page for verification. We automatically verify this page to be a registration page by determining the existence of a (primary) password field as described in Section 6.2.

To improve our registration page detection and verification process throughout our analysis process, we keep track of our success rates. Additionally, we perform manual reviews to find possible missing elements in our verification process to improve accuracy.

6.2 Verifying the registration page

Our approach for determining a page to be a registration page is to look for a password (input) field located in the `HTML` source code. `HTML` password fields are recognized by its input-type for which the `HTML` standard provides a specific `HTML` element[1] as demonstrated below.

```
<input type="password" />
```

Additionally to the page containing a password field, we check and determine the following:

1. The password field is *not* located in the header of the website, as this might be a sign-in form.
2. In case of multiple password fields, we determine the first detected to be the primary field, as elaborated on below.

Detecting the primary password field. There are scenarios where an `HTML` page will have more than one password field. A common scenario is a registration page where a user is asked to enter a password twice, generally for confirmation purposes. An example is shown in Figure 6.2 below from Wiley.com[2] including its

[1] `https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input/password`
[2] `https://onlinelibrary.wiley.com/action/registration`

accompanying HTML in Figure 6.3. In this scenario we consider the first password field (based on the order of the HTML elements) the primary password field.

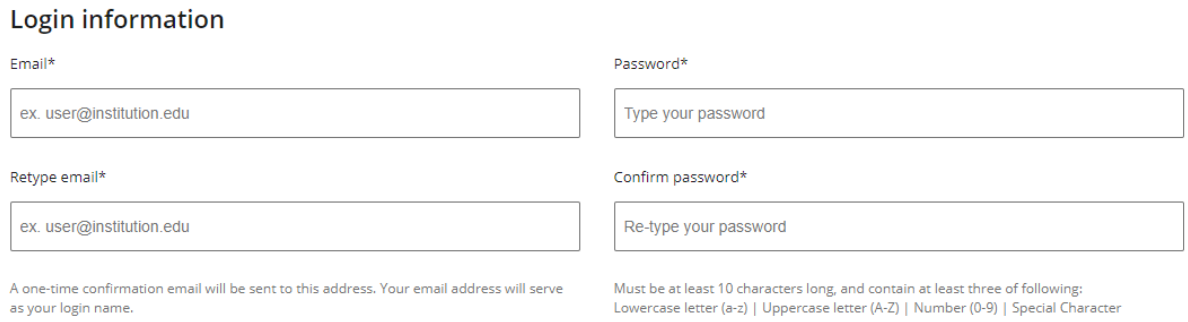


Figure 6.2: User interface of a registration page with two password fields

```
<div class="col-md-6">
  <div class="input-group">
    <div class="label "> </div>
    <input id="login.password" class="pass-hint" aria-labelledby="login.password.label passwordHint" type="password" name="login.password"
    value autocomplete="off" placeholder="Type your password" required="true">
    <span class="password-eye-icon icon-eye hidden"></span>
    <div class="password-strength-indicator" data-min="10" data-max="32" data-strength="4"> </div>
  </div>
  <div class="input-group">
    <div class="label "> </div>
    <input id="login.password2" aria-labelledby="login.password2.label" type="password" name="login.password2" value autocomplete="off"
    placeholder="Re-type your password" required="true">
    <span class="password-eye-icon icon-eye hidden"></span>
  </div>
  <p id="passwordHint" class="note"> </p>
</div>
```

Figure 6.3: Accompanying HTML of the registration page with two password fields

Alternatively websites may have a sign-in password input element in the header of the website. We therefore exclude any password fields that are located in the header-element[3] of a website. Websites may not directly expose their password field on the registration page. In some scenarios a website may first ask for a user email address, and only after submitting the (to be registered) email address or username, the website will ask the user to enter a password. In this initial study, our approach requires a directly reachable password field. Websites with other scenarios can be supported in future research, after establishing the initial validity of our approach.

6.3 Identifying relevant client-side source code

The registration page generally consists of many text strings, HTML elements and JavaScript code lines/blocks. Only a number of these elements are possibly relevant to the password field, and thus the password requirements. In this section we present our approach to identifying each type of element, and how we identify the elements that are worth collecting for the possible inferring of password requirements. The following methods for identification are based on client-side development best practices as well as findings from our exploratory research as performed in Chapter 5.

[3] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/header

Our approach has extensibility and flexibility built-in to allow for future work. To enable extensibility, any used "list" such as password keywords-lists are generically formatted in JSON-format. With this approach, additional keywords can be added or removed, and mapping rules can be customized.

6.3.1 Identifying text strings

Text strings are lines of text that are present throughout the client-side source code. These lines of text may include password requirement hints or instructions, so it is key to identify which lines are relevant. We present two approaches to identify relevant text strings. Both approach are performed regardless if the first approach succeeds or not to identify all relevant text strings.

Based on proximity to the primary password field. Our primary method for identifying password related text strings is by its proximity to the password field. Textual strings located near the password field *or* within the password field element as *generic* `HTML` attributes (e.g. `placeholder` attribute) are likely to contain relevant text. We collect text strings;

- *inside* the password field element (child elements)
- text strings at the password field level (as part of the input element)
- *outward* from the password, until we reach a parent-level that contains another input field.

An example of this is displayed in Figure 6.4 from Fastly[4] where one level *up* from the password field, but before reaching other input field, text is found containing the password requirements.

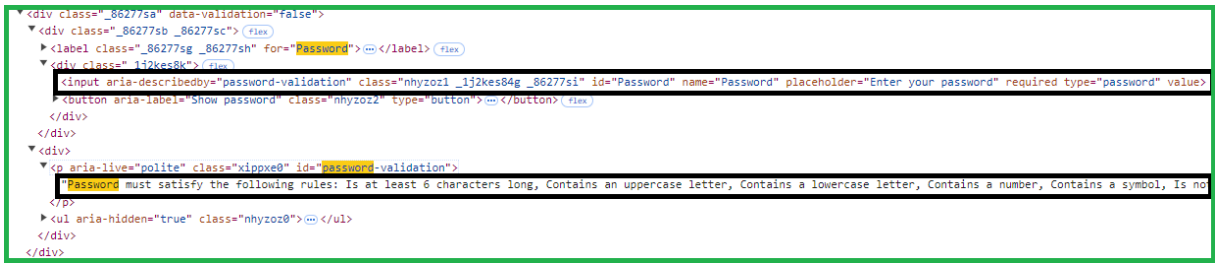


Figure 6.4: Password requirements near the password field, at equal depth from parent level

Based on password keywords. Textual strings may often be located at other places within the source code than near the password field. A common occurrence is that textual strings are centralized at the top or bottom of the `HTML` page or JavaScript block. We filter these text strings by checking if the text contains the word 'password'.

[4] `https://www.fastly.com/signup/`

6.3.2 Identifying HTML attributes

The identification of relevant HTML attributes is a straight forward process by collecting all HTML attributes at the level of the password input field. Any HTML attribute such as `minlength` that is used for validating password requirements is located at the level of the (password) input field. As per example below[5], the password field of the registration page of Intuit.com[6] contains the HTML attribute `maxlength` with value 50.

```
<input
type="password" maxlength="50"
aria-label="Password" aria-required="true"
data-testid="SignUpPasswordInput"
data-bbeid="SignUpPasswordInput"
autocomplete="new-password" name="Password"
id="iux-sign-up-password-input"
value="">
```

6.3.3 Identifying JavaScript

Identifying JavaScript lines and blocks that are relevant to the password field can be performed in two different ways, through inline event handlers, and external event handlers.

Based on inline event handlers. Inline event handlers are located at the same level of the password input field, and therefore can easily be recognized, as the example below demonstrates.

```
<input type="password" onchange="ValidatePassword()">
```

Based on reference of the password field. Alternatively, the password field identifier (`id`, `class`, `name` etc.) can be used to bind a JavaScript function to a field. This method both applies to direct JavaScript function binding (e.g. `onchange`) as displayed below. Any JavaScript referencing the password field is considered relevant scripting.

```
const input = document.getElementById("passwordFieldId");
input.addEventListener("change", ValidatePassword);
```

6.4 Building indicator repositories

Based on the above described method for identifying relevant client-side source code, we can collect and group client-side elements. Per type of client-side element we build a "repository" of collected elements. These repositories can be used to query and subsequently map client-side elements to password requirements. We compose the following three repositories:

[5]This example has been refactored for improved readability by re-ordering attributes and removing unneeded attributes.

[6]`https://accounts.intuit.com/signup.html`

- **Text repository**
  In the text repository we store all relevant text elements that are identified in the client-side source code. In this repository text is stored *per line of text* as a string value to retain its intended textual scope.
- **HTML attribute repository**
  In the `HTML` attribute repository we store all relevant `HTML` attributes elements that are used within the `HTML` password field element. These `HTML` attributes are stored in a Key-Value <string, string> configuration, for example:
  `[Key: minlength] [Value:8]`
- **JavaScript repository**
  In the JavaScript repository we store the type of JavaScript found (inline or reference) including its function in textual representation. This allows for function parsing and interpretation the following steps. The depth level of which we collect the JavaScript code (i.e. depth of the function *tree* or *call stack*) is adjustable and may be set differently based on experience during the collection phase. JavaScript is stored as string value.

Having collected and built the indicator repositories, we can introduce part two of our novel methodology; inferring password requirements from client-side indicators.

Chapter 7

Automatically inferring password requirements from client-side indicators

Our approach to inferring password requirements from client-side indicators revolves around *requirement-to-indicator mapping rules*. Each of these mapping rules has an accompanying *interpretation technique*.

Interpretation techniques are techniques designed specifically for interpreting the data derived from an indicator into the desired information regarding a specific password requirement. In Section 7.1 we further elaborate on the interpretation techniques composed in this study.

Our approach to inferring password requirements is based on three elements;

- A given password requirement.
- A client-side indicator that is suitable (i.e. it *may* hold relevant information).
- An interpretation technique, for a specific requirement/indicator combination.

These three elements combined, result in a password requirement mapping rule.

Different password requirements can vary in terms of data. For example, *minimum length* (a numerical value) is vastly different in terms of data-type than *required character-types* (a group of strings).

The same holds for different client-side requirement indicators. For example text based indicators require a different method of analysis (i.e. 'interpretation technique') than analysing JavaScript functions. This level of specificity results in that specific requirement-indicator-interpretation technique combinations are required in order to successfully infer password requirements.

Lastly, in some cases, different indicators may give contradictory results for the same password requirement. For example, the `minlength HTML` element specifies a minimum length of 6 characters, while a hint text requires at least 8 characters. In such cases, a priority ranking is required to determine which results takes precedent (Section 7.3).

Determining mapping rules. Our approach for determining (new) mapping rules for a password requirement starts with establishing which indicators *can* contain the information needed (i.e. suitability). For example, the `minlength HTML` attribute cannot hold information regarding allowed character type groups.

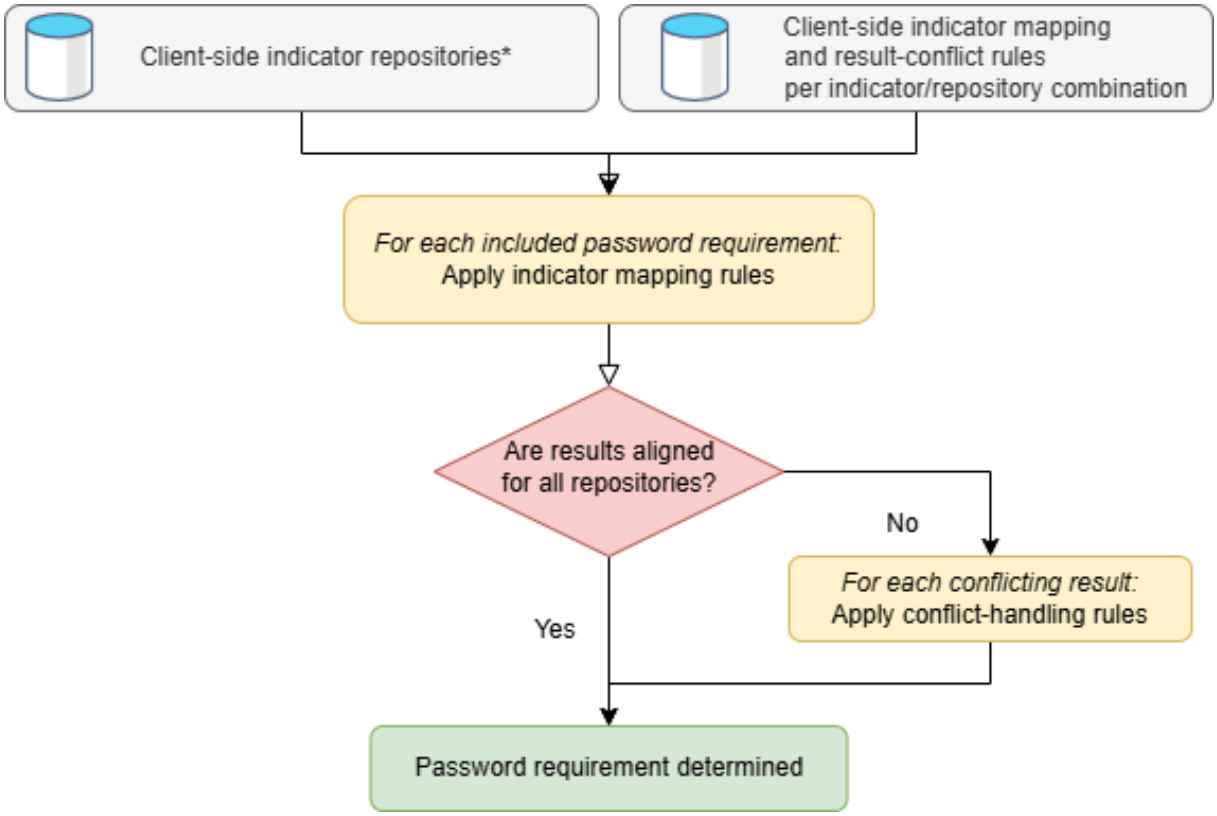
Determining this suitability can be done through manual analysis of websites that use this indicator (e.g. analysing textual strings on websites to determine which information it can hold). Alternatively, when an indicator is based on a standard or publisher that provides technical documentation (e.g. `HTML` elements), that information will generally provide the information scope of the given indicator.

When determined which indicators are suitable for inferring the given password requirement, each indicator must be interpreted by a indicator-specific interpretation technique. Ideally, interpretation techniques are reused. Should such an interpretation technique not yet exist, it must be designed and developed.

A bidirectional approach. In this study we reason from a password-requirement perspective, as this is our specific focus in this study. In reality, it may also occur that within the existing set of password requirements, a new client-side indicator has become suitable. Our process is bidirectional, with interpretation techniques as the essential 'middle part' allowing client-side indicators to be added in the similar fashion as new password requirements are added. By determining suitability and then composing the required interpretation technique.

Conflicting results. As there are possibly multiple suitable indicators for a specific password requirement, conflicting results may occur. In the case conflicts occur, a priority ranking is required to determine which results take precedent. In Section 7.3 we further elaborate on handling conflicting results.

Overview. Having established both mapping rules and conflict handling rules, our approach can be applied to websites and indicator repositories. In Figure 7.1 we present a high level overview of all these steps applied to the client-side indicator repositories as presented in Chapter 6.



* as presented in Chapter 6

Figure 7.1: Automatically inferring password requirements

7.1 Interpretation techniques

Prior to presenting our mapping rules, we first present the interpretation techniques included in this study. To infer the password requirement value (e.g. minimum length of value '8'), from a password requirement indicator, we present interpretation techniques. These interpretation techniques are specific to a requirement/indicator-combination.

7.1.1 Direct result from value

The most straightforward interpretation technique is when an indicator maps directly to a password requirement. An example of this is the `HTML` attribute `minlength` of which its value directly represents the minimum length password requirement. This requires no further interpretation.

7.1.2 RegEx interpretation

RegEx interpretation is notoriously difficult as RegEx expressions are highly customizable. In our approach we have selected a number of RegEx patterns that are, based on developer guides such as Mozilla's developer guide [1], considered standard or rudimentary.

[1] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions`

For this study we interpret the following RegEx notations:

- `a-z` for password requires lowercase.
- `A-Z` for password requires uppercase.
- `0-9` for password requires digits.
- `\w` as a shorthand for `[a-zA-Z0-9]`. i.e. password requires all three classes (lowercase, uppercase and digits).
- `{_@./\#\&+-}` any listed symbol for special characters i.e. password requires special characters.

As RegEx is highly customizable, inferring length requirements is difficult. This is because RegEx's length notation can also refer to a specific part of the string (e.g. minimally 1 uppercase symbol), and not necessarily apply to the whole string. We therefore *only* interpret RegEx's length notation when it occurs only a single time and as the last element in the regex pattern as per examples below:

- `[a-zA-Z]{8,16}$` for length range from 3 to 16.
- `[a-zA-Z]{8,}$` for minimum length of 8.
- `[a-zA-Z]{,16}$` for maximum length of 16.

7.1.3 HTML attribute interpretation techniques

Below we present our interpretation technique for parsing the `passwordrules` attribute and the `onpaste` attribute. It is to be noted that the `minlength` and the `maxlength` attributes are resolved via direct result inference as described in Section 7.1.1.

Parsing the `passwordrules` attribute. The `passwordrules` attribute is a complete password requirement attribute. This means that most password requirements can be inferred from this attribute. Unfortunately this attribute is still in the `HTML` proposal phase, and in our manual website analysis from Chapter 5, we only found this attribute once.

Regardless, for parsing this `HTML` attribute, we can use its easy to understand structure to infer password requirements. This attribute is limited to only length and composition password requirements however. An example of the (string) value of this attribute is:

```
"password-rules": "minlength: 6; required: lower, upper, digit;"
```

We can directly map the following password requirements to the following values:

- {minlength}:{direct value for minimum length requirement}.
- {maxlength}:{direct value for maximum length requirement}.
- {required}:{value requires further parsing}.
  - {upper}:{direct value for required character-type classes-requirement}.
  - {lower}:{direct value for required character-type classes-requirement}.
  - {digit}:{direct value for required character-type classes-requirement}.
- {max-consecutive}:{direct value for 'reject sequential characters'-requirement}.

Using the parsing rules above, we can establish the password requirements based on the value of the `passwordrules` attribute.

Parsing the `onpaste` attribute. The `onpaste` attribute is the direct event handling attribute when a user pastes a value into an input field. In the context of a password field however, this used to be used for blocking password pasting. This is now considered a bad practice as the NIST explicitly recommends the allowance of pasting passwords (e.g. from password managers). In terms of parsing a value from this attribute, the attribute's value contains a JavaScript function and must be interpreted. There are two scenario's in which we conclude that pasting is blocked.

- Inline JavaScript:
  when containing a direct return command, e.g.:

  ```
  onpaste="return false;"
  ```

  or when containing the preventDefault function call, e.g.:

  ```
  onpaste="{preventDefault}"
  ```

- Referenced JavaScript:
  when containing the PreventDefault function call, e.g.:

  ```
  myInput.onpaste = e => {e.preventDefault()};
  ```

  or using an EventListener and containing the PreventDefault, e.g.:

  ```
  addEventListener("paste", {e.preventDefault()});
  ```

In the two scenarios listed above, we consider password pasting to be forbidden. In any other scenario, we consider password pasting to be allowed.

7.1.4 Text interpretation techniques

Our interpretation technique for textual password instructions is based on *text pattern matching*. For each to be inferred password requirement we have composed pattern matching rules that are based on our exploratory findings during our manual website analysis in Chapter 5. This interpretation technique is only applicable to length indicators and character type-class indicators as listed in Table 7.2.

These pattern matching rules are based on common text patterns (i.e. common password instructions). This results in a limitation for our approach as we will only match text patterns that we have explicitly added to our rules. Our mapping rules are *extensible*, so it is possible to add new patterns or add additional languages besides English based text patterns.

In Table 7.1 we present a high level overview of our text pattern rules per password requirement. Within a text pattern, we detect specific characters such as { d+} for digits or fixed words such as {/upper/} for a string containing the word 'upper'. By using the smallest variant of a keyword ('upper' versus 'uppercase') we broaden our matching ability. We use RegEx notation for matching these specific patterns, which are case insensitive.

Password requirement	Pattern
Length	
– Minimum	minimum {d+} at least {d+}
– Maximum	maximum {d+} no more than {d+}
– Range	{d+} and {d+}
Required character type-classes	
– alphabetic	{/lower/} {/letter/} {/alphabetic/}
– capitalized	{/upper/} {/capital/}
– special	{/special/} {/symbol/}
– numbers	{/digit/} {/number/}

Table 7.1: Shortlist of mapping rules for Text pattern matching

Our textual password instruction-repository as presented in Chapter 6.4 provides text strings *per line* of instruction. Hence, we consider each line a separate instruction and pattern match each line according to Table 7.1. The complete list of text matching patterns can be found in Appendix D.

7.1.5 JavaScript interpretation techniques

JavaScript interpretation is a difficult endeavour as JavaScript allows developers to create scripts with much variety and relatively little constraints. Similar to RegEx interpretation, we therefore focus on a small number of key interpretations in this study. In this part of our approach we have already extracted the relevant password validation functions into our JavaScript-repository as described in Section 6.4.

Analyse conditional checks. One of the key methods for JavaScript interpretation is analysing conditional checks. Websites can use JavaScript to enforce password policies by implementing conditional statements that validate user inputs. A (function with a) conditional statement or check, validates specific user input against a condition and results in a `true` or `false`. Alternative a function may also, instead of returning `true` or `false`, simply proceed to the next function or abort (e.g. throw an error[2]).

For determining password requirements by analysing conditional checks, we focus on two determination paths. There are more methods for interpretation JavaScript, but in our study we focus on two rudimental approaches for simplicity and high accuracy. At the end of this section we elaborate more on the limitation of this approach. In order to analyse JavaScript, we first parse the JavaScript function into an Abstract Syntax Tree (AST). This allows us to target conditional statements (i.e. if-statements). An example of AST-parsing can be found on *astexplorer.net*[3]. AST parsers such as Esprima[4] or Acorn[5] are available from the open source community.

In this study we include the following two JavaScript password determination methods:

1. **Using JavaScript's `length`-property**
   Specifically for the password min. and max. length requirement, we know that the `length`[6] property of a JavaScript variable must be used. We can search the AST for this property in combination with a conditional statement. By retrieving the operator (e.g. '<', '>', '==') value of the conditional statement we can determine the min. or max. length of the condition. An example of a conditional statement is shown below.

   ```
   if (password.length < 8) {
   alert("Password must be at least 8 characters long.");
   }
   ```

2. **Using JavaScript's `test`-property**
   In the example below we search for the `test`[7] property to infer a RegEx value, that we can parse using our RegEx interpretation technique. This technique can be used for min. and max. length, as well as required character type

[2] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/throwh`

[3] `https://astexplorer.net/#/gist/e2b13cfd7074c4e5fc2afed54cdb6e3a/6e8dab825553c0e738013826f03129e4a416cf76`

[4] `https://esprima.org/`

[5] `https://github.com/acornjs/acorn`

[6] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/length`

[7] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/length`

classes. Similar to the `length` property, we can use the AST to search for this property and retrieve a possible RegEx value for further interpretation.

```
if (!/[A-Z]/.test(password)) {
  alert("Password must include at least one uppercase
     letter.");
}
```

Check for haveibeenpwned.com. The website (and API) `haveibeenpwned.com` provides a method of comparing a user provided password against a list of breached passwords. By rejection user passwords that have been exposed in a prior breach, the chance of a new (in particular, brute force) breach is lowered. For determining a website's usage of the `haveibeenpwned.com` API we scan for the presence of the `haveibeenpwned.com` endpoint ('api.haveibeenpwned.com') in the JavaScript code.

`haveibeenpwned.com` is not the only provider of this type of service, but is widely regarded as the largest and most authoritative public breach notification and password exposure service available, especially when compared to alternatives such as Breach Detective[8] and Dehashed. [9]

Other providers can be implemented in a similar fashion to `haveibeenpwned.com`. This integration techniques only applies to the password requirement 'Reject known bad passwords/Commonality'.

Check for (pass)word-lists. Similar to the haveibeenpwned.com-technique, we also check for the presence of password-lists in the JavaScript code. If a website uses a hard coded "bad password" list, it likely contains commonly known bad passwords. We therefore don't seek to match an entire list of bad passwords, but only check if one (or more) of the bad passwords in our own list occurs in the JavaScript code. For our own bad password-list we use the top 100 most common passwords.[10] With more time and resources this list can be extended to the top 1000, top 10.000 etc.

This integration techniques only applies to the password requirement "Reject known bad passwords/Commonality".

A remark on the extraction and running of scripts locally. Fundamentally, all client-side source code can be extracted from the website's sources and be activated to run in a testing environment. In this testing environment we can extract the validation function that is responsible for password validation, and *execute* this function locally by passing different passwords to determine its acceptance boundaries. By passing passwords of different lengths, characters etc. it is possible to determine (i.e. reverse engineer) the password requirements that are validated.

In the existing research ([ABG12, ACF+12]) of Alkhalaf et al., this technique was investigated and Alkhalaf et al. achieved some initial success. Their extract-and-run-technique however is very complex in terms of parsing rules, and rigid in terms of applicability to inferring password requirements. During our manual website analysis in Section 5.2.1 we only found 2 websites using JavaScript validation

[8] `https://breachdetective.com`

[9] `https://dehashed.com`

[10] `https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100.txt`

function that was manually discoverable. Therefore, from a limited time and resource perspective, we chose not to include this extract-and-run-technique in this study. We consider this an acceptable limitation to our approach considering the low number of websites found with JavaScript validation logic in our manual analysis (Sec. 5.2.1).

7.2 Mapping rules for password requirements

Mapping rules are the combination of a specific password requirement, a specific client-side indicator, and a specific interpretation technique. However, not all indicators are applicable to all password requirements. Therefore our mapping overview in Table 7.2 is grouped by password requirement, which then only lists the applicable indicators per requirement.

For each password requirement, we present mapping rules to its applicable indicators. Next, we present our mapping rules for the password requirements and indicators that we have included in this study.

a. Length requirement (min. max)

Client-side indicators	*Interpretation techniques*
• HTML attributes	
– `minlength, maxlength`	7.1.1 Direct result from attribute value
– `pattern`	7.1.2 `RegEx` parsing
– `passwordrules`	7.1.3 `passwordrules` attribute parsing
• Text	
– Password instructions & hints	7.1.4 Text pattern matching
• JavaScript	
– Functions containing `RegEx`	7.1.2 `RegEx` parsing
– Function code	7.1.5 Analyse conditional checks
	7.1.5 Extract and run test cases

b. Character type-class requirement

Client-side indicators	*Interpretation techniques*
• HTML attributes	
– `pattern`	7.1.2 `RegEx` parsing
– `passwordrules`	7.1.3 `passwordrules` attribute parsing
• Text	
– Password instructions & hints	7.1.4 Text pattern matching
• JavaScript	
– Function code	7.1.5 Extract and run test cases
	7.1.5 Analyse conditional checks
– Functions containing `RegEx`	7.1.2 `RegEx` parsing

c. Forbidden password commonality requirement

Client-side indicators	*Interpretation techniques*
• JavaScript	
– Function code	7.1.5 Check for haveibeenpwned.com
	7.1.5 Check for (pass)word-lists

d. Allowance of password pasting requirement

Client-side indicators	*Interpretation techniques*
• HTML attributes	
– `onpaste`	7.1.3 `onpaste` attribute parsing

Table 7.2: Overview the mapping rules in this study (the password requirement/indicator/interpretation technique combination).

7.3 Handling conflicting parsing results

In Table 7.2 we presented several password requirements and their mapping rules. For example, the *length requirement (min., max.)* has mappings to the following

password indicators and interpretation techniques:

Length requirement (min., max.) ***Client-side indicators***	***Interpretation techniques***
• HTML attributes	
– `minlength, maxlength`	7.1.1 Direct result from attribute value
– `pattern`	7.1.2 `RegEx` parsing
– `passwordrules`	7.1.3 `passwordrules` attribute parsing
• Text	
– Password instructions & hints	7.1.4 Text pattern matching
• JavaScript	
– Function code	7.1.5 Extract and run test cases 7.1.5 Analyse conditional checks
– Functions containing `RegEx`	7.1.2 `RegEx` parsing

Table 7.3: Multiple suitable client-side indicators for a single password requirement.

The Length password requirement, similar to other requirements, are mapped to *multiple* client-side indicators. This multiplicity results in a higher likelihood the password requirement *can* be inferred, as there are multiple interpretation techniques that can be applied. The applicability of multiple interpretation techniques can strengthen our accuracy, when the results are aligned, when they are not, we must resolve this conflict.

Our approach to handling multiple conflicting password requirement results is by ranking, and prioritizing, the client-side indicators by their estimated level of website enforcement. When we have low trust in that a client-side indicator is actually enforced during the account creation process, it becomes less relevant when determining the password requirements of a website. We specifically approach conflict handling using estimations of enforcement, because through custom programming or scripting, many of the client-side indicators, such as HTML attributes, can be bypassed or simply implemented incorrectly, resulting in improper enforcement.

Prioritizing client-side indicator repositories. Our initial prioritization is based on the source of the client-side indicator, i.e. the type of indicator repository. We have devised the following estimated enforcement ranking for indicator repository.

- **HTML attributes**: High likelihood to be enforced.
- **JavaScript**: High likelihood to be enforced.
- **Text**: Not enforced. Further elaborated on in Section 7.5.

Both HTML attributes and JavaScript provides enforcement in the client-side, either through default web browser behaviour (HTML attributes) or through measurable custom JavaScript validation functions. They thus both take precedent of any conflicting textual indicators.

HTML attributes and JavaScript-indicators require a second level of prioritizing as they both have a high likelihood of being valid client-side indicators. HTML

attributes within HTML forms are 'first to be validated' compared to JavaScript code due to default browser behaviour. This means that HTML attributes essentially block further JavaScript validation execution, *unless* the HTML attributes are more lenient than the JavaScript validation (e.g. HTML attribute allows a value to be submitted, but it is then blocked by a more stringent JavaScript validation). Hence, JavaScript only takes precedent over HTML attributes *if* the resulting requirement is more stringent in JavaScript. Otherwise HTML attributes take precedent.

Prioritizing client-side indicators within single repository. As demonstrated in Table 7.4, within both the HTML attributes and the JavaScript client-side indicator-type, multiple indicators are mapped. In the previous paragraph we presented our priorities among client-side indicator-types (i.e. the different indicator repositories), below we present our prioritization within a specific indicator-type. We have intentionally ordered the client-side indicators in order of priority, where the lowest number (e.g. 1) indicates the highest priority that takes precedent in a conflict over a lower priority (e.g. 2).

Client-side indicators	***Interpretation techniques***	***Priority***
• From HTML attributes repository		
– `minlength, maxlength`	7.1.1 Direct result from attribute value	1
– `pattern`	7.1.2 `RegEx` parsing	2
– `passwordrules`	7.1.3 `passwordrules` attribute parsing	3
• From JavaScript repository		
– Function code	7.1.5 Extract and run test cases	1
	7.1.5 Analyse conditional checks	2
– Functions containing `RegEx`	7.1.2 `RegEx` parsing	3

Table 7.4: Multiple suitable client-side indicators for a single password requirement.

The process for determining priority is two fold. We first determine the level of default enforcement. For example, HTML attributes such as `minlength` and `maxlength` are enforced when used correctly by the browser. The `passwordrules` attribute however is still in the proposal-state, meaning its intended behaviour is not supported by most browsers yet, and thus may be ignored on submitting a HTML form.

When conflicts occur in the first determination approach, we look at the completeness of the interpretation technique related to this indicator. For example, RegEx parsing is difficult and in this study we have a limited RegEx parsing scope for determining only the most common patterns. This means that this interpretation technique is limited in practice, and thus takes lower priority than a interpretation techniques that is of a more 'closed' interpretation technique such as the *extract and run testcases-technique* (Sec. 7.1.5).

There is an exception for some specific password requirements that have multiple interpretation techniques, such as the *Forbidden password commonality requirement* as listed in Table 7.2. This password requirement resolves into a simple *yes* or *no* result. The two interpretation techniques are considered an 'OR' equation, where

only one interpretation technique has to result in a *yes* regardless of the second interpretation technique's result.

7.4 Validation

Our approach to inferring password requirements, as well as our approach to automatically determine client-side indicators in Chapter 6 are novel approaches. Because of this, our approach requires validation to ensure its real world feasibility. For validation, we perform proof-of-concept in Chapter 8. In this proof-of-concept we apply our approach to a number of websites in order to infer their password requirements.

Additionally to the proof-of-concept, we are also interested in validating the validity of specifically textual password instructions of websites. Textual password instructions play a large role in this study as it is the most found client-side indicator during our manual website analysis in Chapter 5.1. For `HTML` and JavaScript client-side indicator, there is (in most cases) browser enforced validation. For textual password instructions there is no default enforcement, hence we do not know to what extend these textual password instructions are actually enforced.

Experiment. To determine the validity of textual password instructions (i.e. is this indicator trustworthy?), we perform a small experiment. In this experiment we re-investigate the websites (as listed in Chapter 5) and tested the validity of their textual password instructions.

We perform the following steps per website under test:

1. **Trigger the showing of textual password indicators.**
   The majority of websites displayed their instructions when clicking on the password field as established in Chapter 5.1. When no instructions are shown, we perform a form submit that is most likely to fail its validation, in order to avoid the creation of an account. We do this by completing the registration form, but for a password using a 'x' value, which in all cases was rejected by server-side validation. This did in all cases trigger the displaying of the website's password requirements or instructions.
2. **Test the rejection bounds of the instructions.**
   For each individual password requirement in the instructions, we test the lower and upper bound. For example, if minimum length is instructed '8' we test, by attempting to submit the form, by using a password with length '7'. We perform a similar action for other displayed password requirements.
3. **Test a single valid password.**
   We finally attempt to submit a *single* successful form in order. We do this by entering a valid password (based on the instructions), on the lower bound. For example, when a website instructs the user for a password with minimal length of 8 and to use a symbol, we enter a password of length 8 with a single symbol. From a password strength perspective the lower bound is most important (as opposed to maximum length for instance), hence we focus on the lower bound accepted password. This approach limits the number of fake accounts created to a single account while still, in combination with the previous step, delivering sufficient value for analysis.

Findings. We performed this experiment on the 100 websites from Chapter 5 and found *no discrepancies* between the server-side validation and the textual password instructions displayed client-side.

We did find a nuance in textual password instructions on websites that requires more elaboration. Out of the 100 websites analysed we found 9 websites that provided textual password instructions, that were actually hints for creating a strong password, and not a requirements. For example, on *Google.com's* registration page, we found the following textual instructions. Each line represents a separate instruction that was found.

> Create a strong password with a mix of letters, numbers and symbols.
> Use 8 characters or more for your password.
> Please choose a stronger password. Try a mix of letters, numbers, and symbols.

These textual instructions actually are a mix of a *minimum length requirement* and tips to help create strong passwords, validated by a strength meter. We have tested this by successfully entering a password only consisting of numbers (e.g. '123456789123456789') that lacks the mix of letters, numbers, and symbols as hinted at, but long enough to pass the strength meter validation. This indicates that password 'strength' is the actual requirement measured as long as minimum length is 8, and the other requirements are just hints. The only concrete password requirement inferrable from this specific case is thus a minimum length of 8. We therefore have to take into account that when supposed requirements are presented, without a concrete 'count' (e.g. 'at least *one* special character'), the requirement may be a tip or hint and not an actual password requirement. In such cases, to retain accuracy, we do not infer password requirements from such textual hints.

Creation of accounts. For this experiment we *did* create an account on the websites under test. Our aim for, and during this study is to minimize our digital footprint on websites and limiting the creation of accounts. We therefore limited the creation of an account only for this particular experiment *and* only a single successful attempt to validate a successful password submit. Because of this minimization, we consider the negative impact on websites minimal and acceptable from an ethical perspective.

Demonstrated validity. This experiment demonstrates the validity of using password instructions on websites as a client-side indicator to infer password requirements.

7.5 Limitations

Our novel approach has a clear main limitation. Because our analysis is based on client-side indicators, we do not measure server-side enforced password requirements. The final step in account creation occurs server-side and thus may include more stringent or different validations due to programming misconfigurations (e.g.

HTML attributes are bypassed using JavaScript) or discrepancies (e.g. textual password instructions are not aligned with possible server-side validations). These discrepancies do impact the accuracy of our approach.

In particular text-based client-side password indicators (i.e. password instructions) are simply 'to be trusted', in particular when there are no other supported client-side indicators such as HTML attributes. However, our experiment in Section 7.4 demonstrates the validity of these password instructions, and its suitability for our novel approach.

Additionally, the approach presented in this study, has a limited number of mapping rules and interpretation techniques. There likely are websites that *do* have client-side indicators but are *not* measurable using our initial set of mapping rules and interpretation techniques. This is in particularly relevant for RegEx and JavaScript interpretation techniques. The accuracy of our approach will improve when interpretation techniques and mapping rules are extended based on a larger number of websites.

Chapter 8

Proof-of-concept: Measuring password requirements through client-side indicators

Our methodology for determining password requirements based on client-side indicators has been theoretical so far. We validate its real-world applicability by way of a proof-of-concept and test its performance in an open-world experiment.

8.1 Overview

We divide the actions performed by our tool into two conceptual steps as follows: *'step 1: finding the registration page*, and *'step 2: inferring password requirements'*.

Step 1: finding the registration page. This step is based on existing research [JKKS20, AL23]. For this proof-of-concept, we only include websites where the password-field is detected *directly* on the first found registration page. Websites that require an email address, personal information, or send a verification code or CAPTCHA *before* displaying the password field, are thus excluded. Our tool also requires websites to be in English.

While this approach suffices to show the viability of our concept, it has many false negatives in regards to found registration pages.

Step 2: inferring password requirements. For our proof-of-concept we focus on the most occurring client-side indicators as determined in Chapter 5.2.1. Our approach to inferring password requirements supports JavaScript parsing, but for this proof-of-concept we focus on the most frequently occurring client-side indicators and therefore exclude JavaScript *validation logic* interpretation.
Step 2 of our tool performs the following actions:

1. Retrieve the website URL from the selection of websites suited for analysis.
2. Complete client-side source code download into memory (`HTML` and both internal as externally hosted JavaScript sources).
3. Build client-side indicator repositories based on source code.
4. Determine password requirements based on repositories and mapping rules.
5. Store resulting findings in datastore for tracking purposes.

8.2 Configuration

We execute our password requirement inferring tool onto the Tranco-list of most popular websites. We consider this list suited for this type of analysis as we also used the Tranco-list for our manual research in this study. As our tool downloads website contents from a website's server, we are very dependent on the speed of the website's server.

We manually monitor progress made during the execution to refine and improve our tool as it runs. Interruption may occur when the machine requires a reboot, or the process gets stuck. To counter this interruption risk, we build-in a timeout of 100 seconds, after which our tool will move on to the next website. We keep track of every website our tool visits. This tracking allows us to restart the process where it left off in case of an interruption that requires manual intervention (e.g. a bugfix is required).

In our proof-of-concept we use a Windows 10 Pro workstation with an Intel Core i9-10900K @ 3.70GHz processor and 64GB of memory. This setup allows the parallel analysis of 8–10 websites. Our tool is written in `C# .NET`,[1] and uses Chrome based Selenium for webbrowser automation. More details regarding the Selenium configuration can be found in Appendix C. We use a Microsoft SQL database for storing the results of the analysis and for keeping track of website visits.

8.3 Results

We divide the results of our tool into two conceptual parts, namely the *'finding the registration page'*-step, and the *'inferring password requirements'*-step. These are two distinct phases in our website analysis, of which our proof-of-concept focusses on the *'inferring password requirements'*-step as this is a new contribution to the research domain.

Results for step 1: finding the registration page. We analysed a total of 43,848 websites with the aim to locate a registration page. This initial analysis *includes* scanning URLs that linked to websites that were not in English, did not have registration options, or were not available or did not have a password field on the registration page.

Our tool detected that 8,254 (18.8%) of websites *did* meet our criteria. These 8,254 websites are considered suitable for the inferring of password requirements in step 2. For the remainder of the websites our tool did not detect a registration page that met the criteria, or detected no registration page at all.

Results for step 2: Inferring password requirements. The focus of our proof-of-concept is on this step, the inferring of password requirements. Our tool uses the pre-composed and filtered list of 8,254 websites from step 1 to attempt password inference. Out of these 8,254 websites, our tool detected one or more password requirement indicators on 3,065 (37.1%) websites.

In Table 8.1 we have listed each password requirement and the percentage of websites, without data filtering, where the tool inferred the specific requirement.

[1] `https://dotnet.microsoft.com/en-us/languages/csharp`

Password requirement	# of websites w/ inference	
Composition		
– Minimum length	1,521 of 8,254	(18.4%)
– Maximum length	1,604 of 8,254	(19.4%)
– Require character-types	284 of 8,254	(3.4%)
Miscellaneous		
– Reject known bad passwords	708 of 8,254	(8.6%)
– Forbid pasting	1 of 8,254	(0.1%)

Table 8.1: The number of websites on which a specific password requirement was inferred

In Table 8.4 we show the percentage of requirements our tool was able to infer in relation to the total number (4,118) of inferred requirements (i.e. which requirements were inferred the most). Figure E.1 and Figure E.2 in Appendix E show the distribution of the min. and max. length requirements found.

Password requirement	# of inferences	
Maximum length	1,604 of 4,118	(38.9%)
Minimum length	1,521 of 4,118	(36.9%)
Reject known bad passwords	708 of 4,118	(17.2%)
Require character-types	284 of 4,118	(6.9%)
Forbid pasting	1 of 4,118	(0.1%)

Table 8.2: Inference count per requirement in relation tot he total of 4,118 inferred requirements

Filtering out outliers likely to be false positives. During manual validation of the results, we found outliers that were unjustly marked as a specific requirement. For instance, a minimum length requirement of 4 was detected while in reality it was a PIN-code and not a password. Incidently we also detected invalid values such as a minimum length of zero. To determine compliance *and* non-compliance levels accurately, we create a data filter for both the minimum and maximum length requirement. These filters are based on manual validation of the data. This issue is less prevalent for the other requirements.

- For minimum length: we require detected length > 4 AND detected length < 32 to be considered valid.
- For maximum length: we require detected length > 14 to be considered valid.

Explicit versus implicit (non-)compliance. We use the term *explicit* to indicate that we have inferred a certain requirement and thus can conclude with high accuracy whether the inferred requirement is compliant or non-compliant. *Implicit*

compliance is used when no requirement is found, but this is inline with a password standard. For example when the NIST standard requires *no character-types* must be enforced, a website not exhibiting character-types in their indicators is compliant. Implicit compliance has significantly less accuracy as this might also be a false negative (i.e. our tool did simply not detect it).

Compliance to individual requirements. Analysing some individual requirements allows **explicit** compliance, which refers to the scenario where we *are* able to infer a password requirement that *is* compliant with a standard. For example, for *minimum length*, after data filtering, we were able to infer this requirement on 1,143 websites. of which 52.0% of these inferred requirements is determined to be compliant with the BIO/BSI/NIST standard. Thus for the remainder, 48.0%, we can determine explicit non-compliance to the BIO/BSI/NIST standard. We list the results in Table 8.3.

Password requirement	**# of req. inferred**	**BIO**	**BSI**	**NIST**	**OWASP**
Composition compliance					
– Minimum length	1,143	52.0%	52.0%	52.0%	7.6%
– Maximum length	1,432	×	×	68.4%	68.40%
– Require character-types	284	100.0%	100.0%	–	–
Miscellaneous compliance					
– Reject known bad passwords	708	×	100.0%	100.0%	100.0%
– Forbid pasting	1	×	–	–	–

×: password requirement not included in standard or guideline
– : unable to establish explicit compliance

Table 8.3: Percentage of explicitly compliant requirements per password standard

Determining the complete compliance of websites to a password standard combines *explicit* compliance for length requirements, and *implicit* compliance for required character-types.

Standard	**# of compliant websites**
NIST	490 of 8,254 (5.9%)
BIO/BSI	104 of 8,254 (1.3%)
OWASP	61 of 8,254 (0.7%)

Table 8.4: Complete compliance per password standard for composition requirements

8.4 Analysis

When analysing the results of the found password requirements, the strong presence of the minimum length requirement as opposed to other requirements across websites stands out. Detecting the blocking of password pasting stands out as it has the lowest detection frequency. This *is* inline with the current standards and websites *not* adopting the password-paste-blocking practice is considered a safe and modern practise.

The results of the proof-of-concept demonstrate that inferring password requirements based on client-side indicators is technically feasible, but not without limitations. While common requirements such as minimum length and character types were often correctly inferred, more complex (e.g. JavaScript-based validation) or less explicitly stated (e.g. rejecting sequential characters) requirements proved more difficult to detect. We expect that with further improvement of both the approach (e.g. more and improved mapping rules) as well as the technical implementation, the accuracy and number of websites suited for this approach will increase.

Discussion. We were able to determine explicit compliance and non-compliance of *some individual* requirements, in particular minimum and maximum length. This is were our approach seems to shine more than when determining complete website compliance which includes requirements that work by omission (e.g. *not* requiring character-types). When *not* finding specific requirements, it is difficult to establish whether this is a case of non-compliance versus just not inferrable from the client-side.

In regards to the results, we did expect to see more indicators that demonstrate the blocking of password pasting, but as this is not compliant with NIST we are pleased with this, from a security perspective. We also did expect more presence of the 'required character-types'-indicator. But similar to the allowance of pasting, this may be a requirement that is being deprecated as more modern, up-to-date, password standards are implemented across websites.

We compare our results to that of Alroomi and Li [AL23], who performed end-to-end testing including server-side analysis using a 'account creation'-approach. We see a similar pattern in compliance levels for the *require character-types*-requirements, but for *minimum length* Alroomi and Li detected noticeable lower levels of compliance (25% versus 52% for NIST in our study). Alroomi and Li did analyse 2.3 times more websites than in our analysis (20K versus 8.5K in our study). We also detected significantly less password-pasting-blocking practices.

Overall we do think more websites need to be analysed, but due to time constraints we were limited the the current results. In particular finding suitable registration pages took more time than we initially expected. The password requirement inferring-process itself was not a performance bottleneck.

Part II

Private websites internal to an organisation

Chapter 9

Methodology

In order to answer the main research question and its sub-questions, a research strategy must be devised. This part focusses on websites in the private-organisational context and requires a different approach than public websites which we investigated in Part I. We therefore focus on organisational and business research strategies (e.g. surveys and case studies) as described by Saunders [Sau12] in *Research methods for business students*.

For ease of readability we refer back to the research questions as introduced in Chapter 1.

Main research question

> **To what extent are password input requirements of websites compliant with password standards?**

In this part of this study, we investigate private-organisational websites, of which the registration page is generally shielded from the public. We investigate sub-question **RQ 3** from the sub-questions below:

RQ 1. How to automatically infer password input requirements from public websites, at scale, in an ethical manner?
RQ 2. To what extent are public websites compliant to the password input requirements of password standards?
RQ 3. To what extent are private-organisational websites compliant to the password input requirements of password standards?

Selection of methodology and alternatives. To determine password requirement compliance of private-organisational websites, multiple strategies are available. The strategies we considered and selected are listed below:

- **Interviews**
  In this study, we emphasize research at scale, meaning a large number of interviews must be performed in order to collect sufficient data points. Therefore we consider interviews not a suitable strategy for this study due to time constraints.
- **Desk research**
  Existing research into organisational security is a large field of study. However, research into website password policies within organisations specifically is a

niche field that to our knowledge does not have a large literature base yet. As existing research and publicly available information is limited, performing desk research in the broad sense will aid in acquiring additional context, but is less suited for actual data collection.

- **Case study**
  A case study into a specific type or organisation requires publicly available information. A quick search-engine search for publicly available password policies has led us to educational institutions. These organisations appear to frequently publicize their internal password policies. This makes this specific sector suited for a case study as this sector is also considered at risk for cyber attacks.[1] Therefore, we can delve further into the publicly available policies of educational institutions, in order to collect password policy data. We consider this a limited, but suited, approach for determining compliance across private-organisational websites.
- **Survey**
  A survey allows us to request internal information from organisations, while being efficient with our limited time. Surveys allow for a larger scale than interviews, which makes it suitable for this study. As opposed to the before mentioned case study, we can use surveys for a sector that has little to no public information in regard to their policies. A survey can partially overlap the case study, as both aim to discover password policies. Surveys however can add additional context by adding sector specific questions, or questions that provide broader insights into the matter that public documents might not disclose. When looking at which sector to survey, we found several sectors that are considered at high risk.[2] From an initial outreach to gauge survey willingness, we found that municipalities in particular are open to share information regarding their policies. We therefore consider surveying municipalities, part of, a suitable approach for determining organisational compliance to passwords standards.

In order to determine to what extent private-organisational websites adhere to password standards we (1) analyse both publicly available organisational policies via a case study on the educational sector, and (2) analyse private policies of municipalities by requesting information through an organisational survey.

[1] https://www.criticalstart.com/cybercriminals-attack-vectors-within-the-education-sector

[2] https://www.cisecurity.org/about-us/media/press-release/new-report-highlights-critical-infrastructure-threats-and-the-role-of-state-and-local-government-c

Chapter 10

Case study: publicly available password requirements of private-organisational websites

In this case study we investigate educational institutions and publicly available polices regarding passwords.

10.1 Case study design

Selection of organisations. We have searched online for password policies that are publicly available in different sectors such as banking, healthcare and the education sector. We found that that the education sector, in particular universities, publish their password policies publicly available for review. Based on this public availability we have selected universities for our case study. For this case study we have limited the scope to only Dutch, applied and academic, universities as both researchers are Dutch.

Universities and availability of password policies. We have composed a list of universities in the Netherlands for this case study. The Netherlands has a total of 14 universities[1] of which 9 have public password policies.

We have listed all 14 universities and indicated whether their password policy is public, private or derivable. Derivable means that we found information on the organisation's website that is not directly a password policy, but of this information we can derive (parts of) the organisation's password policy. An example of a derivable password policy is Tilburg University's webpage 'Tips for creating a strong password'.[2] This webpage mentions requirements such as the University's requirement to create a new password every year (i.e. the password expiration requirement).

Despite the Open Universiteit not having their policies publicly available, we are able to determine its password requirements as this study is part of the Open Universiteit's curriculum and we have access to its internal (student) environment. We also include universities of *applied sciences*, for which we refer to Appendix F.

[1] https://nl.wikipedia.org/wiki/Universiteiten_van_Nederland

[2] https://www.tilburguniversity.edu/nl/over/gedrag-integriteit/privacy-en-security/informatiebeveiligingsbeleid/wachtwoord

Academic University	**#students**	**Public?**
Universiteit van Amsterdam	41,143	√
Universiteit Utrecht	40,000	√
Erasmus Universiteit Rotterdam	37,051	×
Rijksuniversiteit Groningen	34,401	√
Universiteit Leiden	34,000	√
Technische Universiteit Delft	27,080	×
Radboud Universiteit Nijmegen	24,633	√
Vrije Universiteit Amsterdam	24,517	×
Universiteit Maastricht	22,383	√
Universiteit van Tilburg	19,497	partial
Open Universiteit	16,940	×
Wageningen Universiteit	13,275	√
Universiteit Twente	12,038	√
Technische Universiteit Eindhoven	11,985	×

Table 10.1: Public availability of password policies of academic universities.

10.2 Results

The findings of each university's password policy is presented in Table 10.2. In this table we present the password length requirement, the character classes and how many of these classes are required (indicated as 'Req.'), allowance of spaces (indicated as ' ') and whether 'bad' passwords are blocked. Bad passwords are passwords using the email address as a password, or using a commonly breached passwords.

University	Length	Character classes	Req.	' '	disallowed
Academic					
UvA	12+	[a-z], [A-Z], [0-9], [sym]	4 of 4	–	Y
Utrecht	10 –32	[a-z], [A-Z], [0-9], [sym]	3 of 4	–	Y
Groningen	14+	[a-z], [A-Z], [0-9]	3 of 3	–	Y
Leiden	12 –50	[a-z], [A-Z], [0-9]	3 of 3	–	Y
Nijmegen	10 –40	[a-z], [A-Z], [0-9], [sym]	4 of 4	–	Y
Maastricht	15+	[a-z], [A-Z], [0-9], [sym]	3 of 4	Y	Y
Tilburg	8+	–	–	–	–
Open Universiteit	8+	[a-z], [A-Z], [0-9], [sym]	4 of 4	N	Y
Wageningen	8+	[a-z], [A-Z], [0-9], [sym]	3 of 4	N	Y
Twente	14 –60	[a-z], [A-Z], [0-9]	3 of 3	N	–
Applied Sciences					
Amsterdam	12+	[a-z], [A-Z], [0-9], [sym]	4 of 4	–	Y
Fontys	8+	[a-z], [A-Z], [0-9], [sym]	3 of 3	–	Y
Rotterdam	8+	[a-z], [A-Z], [0-9], [sym]	4 of 4	–	–
Utrecht	10+	[a-z], [A-Z], [0-9], [sym]	3 of 4	–	–
Avans	8+	[A-Z], [0-9], [sym]*	3 of 3	–	Y
Groningen	16+	[a-z], [A-Z], [0-9], [sym]	3 of 4	–	Y
Windesheim	16+	–	–	–	Y
ArtEZ	12+	[a-z], [A-Z], [0-9], [sym]	4 of 4	–	Y
De Kempel	16+	–	–	–	–
Codarts	8 –16	[A-Z], [0-9], [sym]	3 of 3	–	Y

General: Y: explicit yes, N: explicit no, –: not specified

Table 10.2: Password policies of universities

10.3 Analysis

As some found policies do not specify *all* password requirements from Table 10.2, we only deem a university non-compliant if it's policy *explicitly* deviates from the password standard we compare it to. For example, if the space-character is required by NIST, and there is no mention of allowing spaces in a universities policy, we do not consider the policy in-compliant. This only applies to the 'minor' requirements *allow spaces* and *reject bad passwords*. In case more requirements are missing we exclude this university as a whole from our evaluation.

To this end, we analysed 20 publicly available password policies of Dutch (applied sc.) universities and found that 17 out of 20 (85%) universities are compliant with the Dutch BIO 1.0 and German BSI standard. For NIST we found that only 3 out of 20 (15%) are compliant, and for OWASP only 2 out of 20 (10%) are compliant. This non-compliance to NIST and OWASP is primarily from universities requiring *character-types*, which deviates from the NIST and OWASP standard.

Overall we found that the majority of universities in this analysis have a commonly shared view on password policies with minor differences among each other. For example 14 out of 20 analysed policies explicitly mention having validation against bad passwords (e.g. not including email-addresses, names, or previously used passwords).

Limitations. Our analysis of universities has several limitations. As we only analysed the *publicly* available password policies of universities, we cannot rule out the existence of secondary password policies for specific applications (e.g. different policies for students versus staff). In our analysis we did not verify the level of enforcement of these password policies, thus the actual policy enforcement may differ from the publicised policy. Despite not having tested enforcement levels, we do consider the publicly available password policies a strong indicator of the organisation's view on password security.

Chapter 11

Survey: information security professionals on password policies and processes

In this case study we investigate municipalities and their approach to password policies for their private-organisational websites.

11.1 Survey design

We survey civil servants at Dutch municipalities responsible for their organisation's password policies. We also survey them about additional password related information (e.g. the basis for these policies) and map the results to current password standards. We use LinkedIn for both searching the right civil servants based on job title (e.g. CISO, security officer) and for initiating contact through LinkedIn messaging.

11.1.1 Implementation and survey composition

Our survey is created and distributed using LimeSurvey[1], a popular GDPR compliant surveying tool promoted by the Open Universiteit as the default surveying solution. Our survey consists of the following parts:

- **Part 1: About your organisation.**
  Questions in regards to the organisational composition, such as its staff size and cybersecurity team composition. These questions provide additional context which allow us to possibly detect patterns between password policy-types and organisational characteristics.
- **Part 2: Your organisation's password policy.**
  This survey section contains questions specifically regarding the password requirements. For example; What is the password composition policy of the organisation?
- **Part 3: The reasoning behind your organisation's password policy.**
  This survey section contains questions about why an organisation's policy is the way it is. For example how often the policy is re-evaluated.

[1] `https://www.limesurvey.org/`

The aim of this section is to gauge whether the organisation has a complete password policy based on up-to-date standards or insights and a structure including re-evaluation. *Not* having a re-evaluation structure may leave organisations at risk. For example due to an outdated password policy.

Our survey mostly consists of open ended questions and open answers, allowing respondents to provide more context to their answers. With 12 questions our survey is relatively short, aimed at achieving high a number of respondents. The survey can be found in Appendix G.

11.1.2 Privacy considerations

Despite that we consider the information we are interested in *not* highly confidential, organisations may be unwilling to share internal policies. For this reason we anonymize the results of our survey and also emphasize this during our outreach in order to increase the number of respondents. Based on the results and analysis of this study, no organisation can be identified or linked to specific answers.

11.2 Results

Background of respondents & general statistics

A total of 33 respondents have participated in this survey. Some participants communicated that they left certain specific questions blank, to avoid, to what they considered, high risk organisational details regarding password policies. Therefore not all questions may amount to the total number of respondents that participated.

Role	**# of respondents**	**%**
CISO	28	85%
(Information) Security officer	2	6%
Compliance/privacy/risk officer	2	6%
Information advisor	1	3%

Table 11.1: Roles of the respondents

organisation size (staff count)	**# of organisations**	**%**
1-500	19	58%
501-1000	7	21%
1001-1500	5	15%
1501-2000	1	3%
2000+	1	3%

Table 11.2: Size of the organisation in terms of headcount

Team size	# of organisations with team size	%
1-5	27	82%
6-10	3	9%
11-15	0	0%
16-20	2	6%
21-25	0	0%
26-30	1	3%

Table 11.3: Size of the organisational security team (headcount)

Answers to password policy questions

100% of respondents answered that their organisations allows users to create their own passwords. This is inline with the specific standard (BIO, section 9.4.3) this group of organisations is required to adhere to.

- The most occurring minimum length requirement was 12 characters. The complete distribution is presented in Figure 11.1
- 22 out of 33 (66%) answered that their policy requires specific character types such as alphabetic, capitalized, numbers or symbols.
- 1 respondent (3%) added that their required character-type requirement is omitted when a password has a length of 20 characters or more, inline with BIO's recommendation.
- 8 out of 33 (25%) explicitly allow spaces.
- 9 out of 33 (28%) reject sequential character patterns (e.g. 123, aaa).
- 9 out of 33 (28%) reject known bad passwords.
- 1 respondent (3%) allows non-ASCII characters.
- 2 out of 33 respondent (7%) explicitly state that previously used passwords (up to 5 historic passwords) are not allowed to be re-used.
- All respondents stated that these policies are enforced through their Single Sign-on solution. One respondent also stated that these password requirements are also included the the procurement process for new software vendors.

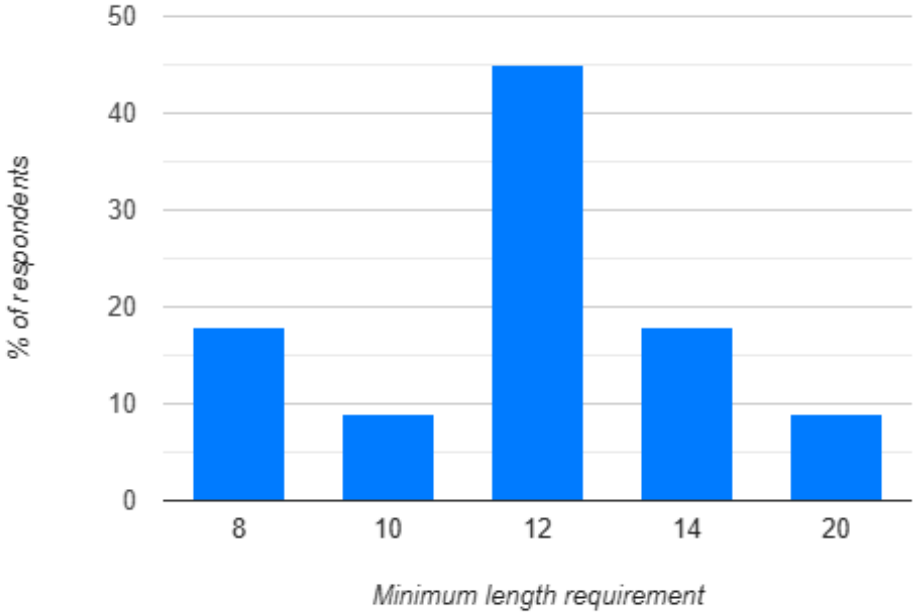


Figure 11.1: Distribution of minimum length requirement across respondents

Additional security layers All respondents stated that 'MFA is always required', one respondent added that in some cases legacy systems do not support MFA and is therefore not enabled.

Resetting a user password is in 44% of cases possible via a self-service portal or using the help-desk's assistance. In 27% of cases resetting a password can *only* be done by contacting the help desk.

A single respondent stated that resetting a password requires an in-person meeting for ID verification.

Password expires	**# of respondents**	**%**
Never	3	14%
After 40 days	1	5%
After 90 days	5	23%
After 180 days	10	45%
After 365 days	1	5%
After unspecified # of days	2	9%

Table 11.4: Password expiration policies

Context of password policies

Password standard	# of respondents	%
Local gov. guidelines (BIO, NIS2)	18	82,0%
NIST	1	4,5%
Microsoft	1	4,5%
Combination (NIST/NCSC/Microsoft)	1	4,5%
Combination (not specified)	1	4,5%

Table 11.5: Basis on which the password policy is based

Policy re-evaluation	# of respondents	%
Yearly	18	82%
Every 2 to 3 years	2	9%
No scheduled evaluation (e.g. when staff decides themselves)	2	9%

Table 11.6: Password policy re-evaluation cycle

Comparing against password standards All municipalities are compliant with the min. and. max. length of the BIO, BSI, NIST and OWASP standard. 22 out of 33 (66%) are *not* compliant with NIST and OWASP as the respondents' policies explicitly require specific character-types. Oppositely, these same municipalities *are* compliant with the BIO and BSI standard as a whole.

As some respondents (33%) left the question 'which character type classes are required' blank, we cannot draw a conclusion for these municipalities. This lack of response may mean 'no character type classes required' versus simply not answered. We therefore do not conclude non-compliance for these municipalities.

11.3 Analysis

The results demonstrate a consistent compliance level to local standards (BIO and BSI) over international standards (NIST and OWASP). This is inline with expectations as local standards take precedent over international standards. The discrepancy between the BIO/BSI and NIST/OWASP is understandable, as an organisation cannot be compliant with both standards simultaneously due to its conflicting *required character types*-requirement.

Many respondents rely on supplier defaults or compliance obligations rather than internal evaluation, which may result in outdated policies. Noticeable is that there is no consistency in how and when municipalities review their policies. This may be due to an over-reliance on the BIO standard.

Discussion. We expect the discrepancy between BIO/BSI and NIST/OWASP to decrease as the expected 2025 updates for these standards are more aligned with each other. Some civil servants indicated that they would rather approach password policies more actively (for example by following the NIST updates), than to simply adhere to an outdated, as they considered it, BIO standard.

We did encounter difficulty to get a high number of respondents. Some civil servants we reached out to declined to cooperate due to security and privacy concerns. A small number of civil services required a phone and/or video call to verify the researcher's intentions, after which the survey was completed. This demonstrates the perceived risk of sharing this type of internal policy. Overall we do consider the number of respondents sufficient to establish a baseline for this study, as it includes different municipality sizes and civil servants job roles.

Chapter 12

Conclusions

In this study we investigated password input compliance to the password standards at scale, both on public and private-organisational websites. This allows us to formulate an answer to the research question '*To what extent are password input requirements of websites compliant with password standards?*', both at an overall level and in more detail, by answering the sub-questions.

Our main conclusion is that overall, both public websites and private organisational **websites exhibit significant non-compliance** with respect to international password standards. **Public websites** generally have to adhere only to self-imposed password regulations, leading to no password policy consistency. In contrast, we found that **private-organisational websites** do aim to adhere to password standards. However, we also found they are slow to incorporate updates to standards, leading to outdated policy implementations.

Answers to the sub-questions provide more detail:

1. **RQ 1: How to automatically infer password requirements from public websites, at scale, in an ethical manner?**
   We argued that creating accounts may impose an undue burden on website owners. Thus, to minimize impact to site owners, analysis ought to be confined to the client-side. This raised the questions whether such a client-side-only analysis can suffice to automatically infer passwords, and whether this analysis can be done at scale.
   We showed client-side-only analysis suffices by designing a password requirement inference method that relies only on client-side analysis. Our approach locates registration pages, extracts client-side indicators (`HTML` attributes, textual instructions, and JavaScript code), and maps these indicators to standardized password rules.
   We implemented this design in a proof-of-concept analysis tool and tested this tool on 8,254 sites, which demonstrates the feasibility of applying this approach at scale.
2. **RQ 2: To what extent do public websites adhere to password requirement compliance standards?** Applying our client-side analysis approach by way of proof-of-concept to Tranco-list, we analysed over 8,254 websites for password input requirements and compared these findings to several password guidelines. We found that 52% of websites that allowed for minimum length inference adhere to the NIST minimum length requirement, versus 48% were

non-compliant.

3. **RQ 3: To what extent do organisations adhere to password requirement standards in private systems?** Based on the performed case study and survey in two different organisational sectors, we found that most organisations *do* implement one or more password standards or guidelines, but these are often outdated and no longer aligned with the current (e.g. NIST) standard(s). When comparing our findings to the current standards, we found that 100% of municipalities and 85% of (applied) universities are compliant with the BIO 1.0 standard.

12.1 Reflection

In this study we set out to determine the password requirements and adherence to compliance of websites, and organisations, using an approach that had yet to prove its feasibility. Our research is motivated by the desire to improve the security of websites and organisations, by first collecting data and insights in how these subjects fare compared to the current standards.

The design of our approach presented in this study was driven by the requirement to be low-intrusive to websites. We successfully designed an approach and proof-of-concept that has proven the feasibility of the low-intrusiveness philosophy that we started with. We did however encounter lower applicability of our approach in terms of the number of websites that are suited for our approach, than traditional methods using account creation. In particular, an in-depth approach for JavaScript analysis lacks in this study due to the dynamic nature of the language, making JavaScript interpretation difficult. We did establish in our manual website discovery analysis that the number of websites using JavaScript libraries for password validation was relatively low. Thus we consider the impact on missing JavaScript analysis limited. It remains difficult to truly determine a website's password requirements when not being to analyse its server-side validations (i.e. performing an end-to-end analysis). But, when performed at scale, we do consider to have demonstrated the value of a low-intrusive approach, by collecting password compliance data across a landscape of different websites.

Ethical aspects of this study. The ethical aspects of the approach presented in this study were key to our research direction. The key ethical-driven aspect is that we avoided the creation of accounts on websites in an automated manner at scale. We managed to do this successfully. During our study however, we did create accounts on websites in a manually performed manner, for discovery and validation purposes. This demonstrates that a strict no-account creation study in all aspects is likely unfeasible, especially in initial studies where validation and a proof-of-concept is of high importance. In the context this study as a whole, the number of accounts created *manually* for these purposes were minimal, and we do not consider this a weakening of our core philosophy of low intrusiveness.

12.2 Future work

In general, extending the dataset used in this study, will contribute to a broader insight into the use of password composition policies. This can be done by for example analysing more websites, organisations in different sectors and including more password standards. Capturing more meta-data from websites and organisations for a more detailed classification (e.g. country, type of website, frameworks used, commercial/non-commercial), may also provide more context to the resulting findings. We list several research directions that may provide additional contributions;

1. **Specific improvements to our technical approach.** Our study's approach for JavaScript analysis is technically limited and future research regarding a more in-depth approach to JavaScript analysis may contribute to this research domain. Additionally, the broadening of the number of client-side indicators that our study supports, as well as more languages in order to analyse more websites, may be of value. This study's approach to text analysis for password instructions/hints was based on fixed-string interpretation, extending this approach or redesigning this with for example AI-based interpretation, can likely increase the number of websites suited for analysis and its accuracy.
2. **Different approaches to determining password policies.** In this study we investigated websites though source code analysis, and organisations via surveys and desk research. Other approaches to determine the password policies of websites and organisations are conceivable and may yield valuable insights. Other approaches for example are analysing source code open-source CMS frameworks (e.g. WordPress) and analysing password validation libraries on for example GitLab. These frameworks likely have default implementations of password policies which can be compared to the current standards. In a broader sense, as our study did not include server-side analysis, it may be valuable to investigate open source projects in general and specifically their password code, both client- and server-side.
3. **Password policies in different domains.** This study is limited to public websites and organisations. Investigating password policy compliance may be interesting in different (technical) domains such as IoT-devices, hardware devices such as (consumer) routers from ISP's that come with standard passwords, and other devices with significant risk for password based attacks.
4. **A less intrusive approach to (website) analysis for other purposes.** The philosophy of valuing a low intrusive approach to analysis is what sets this study apart from existing, related research. This approach of low intrusiveness can be applied to other types of analyses-based research, where traditionally automated approach with user simulation are the de-facto standard. For example, research into different types of website authentication handling where registering accounts is not truly required and data can be collected in a different manner.

References

[ABG12] Muath Alkhalaf, Tevfik Bultan, and Jose L. Gallegos. Verifying client-side input validation functions using string analysis. In *ICSE*, pages 947–957. IEEE Computer Society, 2012.

[ACF+12] Muath Alkhalaf, Shauvik Roy Choudhary, Mattia Fazzini, Tevfik Bultan, Alessandro Orso, and Christopher Kruegel. Viewpoints: differential string analysis for discovering client- and server-side input validation inconsistencies. In *ISSTA*, pages 56–66. ACM, 2012.

[AL23] Suood Alroomi and Frank Li. Measuring website password creation policies at scale. In *CCS*, pages 3108–3122. ACM, 2023.

[CJKR21] Stefano Calzavara, Hugo Jonker, Benjamin Krumnow, and Alvise Rabitti. Measuring web session security at scale. *Comput. Secur.*, 111:102472, 2021.

[GHS21] Eva Gerlitz, Maximilian Häring, and Matthew Smith. Please do not use !?_ or your license plate number: Analyzing password policies in german companies. In *SOUPS @ USENIX Security Symposium*, pages 17–36. USENIX Association, 2021.

[JKKS20] Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and Marc Sleegers. Shepherd: a generic approach to automating website login. 2020.

[KSK+11] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: measuring the effect of password-composition policies. In *CHI*, pages 2595–2604. ACM, 2011.

[LSN22] Kevin Lee, Sjöberg Sten, and Arvind Narayanan. Password policies of most top websites fail to follow best practices. In *SOUPS @ USENIX Security Symposium*, pages 561–580. USENIX Association, 2022.

[MKV17] Peter Mayer, Jan Kirchner, and Melanie Volkamer. A second look at password composition policies in the wild: Comparing samples from 2010 and 2016. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 13–28, Santa Clara, CA, July 2017. USENIX Association.

[Sau12] Mark Saunders. *Research methods for business students / Mark Saunders, Philip Lewis, Adrian Thornhill.* Harlow, England New York : Pearson, Harlow, England New York, 6th ed.. edition, 2012.

[SHPS17] Tobias Seitz, Manuel Hartmann, Jakob Pfab, and Samuel Souque. Do differences in password policies prevent password reuse? CHI EA '17, page 2056–2063, New York, NY, USA, 2017. Association for Computing Machinery.

[VHS17] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. Measuring login webpage security. In *SAC*, pages 1753–1760. ACM, 2017.

[WW15] Ding Wang and Ping Wang. The emperor's new password creation policies. Cryptology ePrint Archive, Paper 2015/825, 2015. `https://eprint.iacr.org/2015/825`.

[Y+04] Mikko Ylén et al. Centralized password management in a global enterprise. *TO FIND!*, 2004.

Appendix A

Ethics review for automated website analysis

B. APPENDIX: ETHICS REVIEW REQUEST

NOTE: This appendix contains the document that is sent to the Open University's Ethical Research Board for Informatics (ERBi) for research method review.

Subject: Requesting research method review

Dear members of the Research Ethics Committee,

In the context of composing a research proposal in the field of IT security and in particular website security, I am requesting an ethics review of a proposed research method.

Research context The number of cybersecurity incidents and data breaches is increasing year over year. Therefor, I am researching a key aspect of internet security, which is the strength of password authentication mechanisms on public websites. This strength depends in part on the strength of the used password. To that end, many sites pose password requirements. These password requirements can vary between websites and its enforced can take many shapes.

Research method for review Part of our research aim is to measure the state of password requirement enforcement. To this end, the various methods to discover the level of password requirement enforcement are of particular interest to us.

Password requirements can be enforced during construction of the password, typically in the front-end of a website (e.g. using JavaScript or HTML-constraints clientside), and after submission of the password, in the back-end of a website (e.g., PHP-code serverside). We intend to measure how (and to what level) websites enforce password requirements by submitting a variety of passwords to both front-end and back-end.

Measurement of front-end enforcement can be performed in isolation: front-code code runs clientside and can be investigated or executed offline. However, the back-end code is not available. Therefore, measuring how the back-end treats a password requires calling the back-end (e.g. directly via HTTP POST requests). In our current methodology, multiple such calls per back-end are typically required to determine password strength enforced by the back-end. (The main exception: only one call is needed in case there is no back-end enforcement.) This approach thus cannot avoid interaction with production systems, hence this request for ethical advice.

The exact research method details of discovering back-end levels of password requirement enforcement are not yet concrete. We foresee that the final method may incorporate the creation of multiple free-to-register accounts per back-end – while automated account creation is not trivial, it is easier than automated account creation AND automated password updating of said newly-created accounts. The latter is not feasible within the confines of this project. (It is not clear whether automated account creation will be feasible either, but it is an option we are considering.)

In reference to the Menlo Report The Menlo Report[74] provides guidance for evaluating ethical dilemma's in CS security research. Below, we apply the Menlo Report's guidelines concisely to our case:

1. **Respect for Persons.** No person or personal data is used in this research method.

[74] https://www.dhs.gov/sites/default/files/publications/CSD-MenloPrinciplesCORE-20120803_1.pdf

2. **Beneficence.** This research method might do harm:

    - Overload server: there is a risk that calls to the back-end can overload the server. This is mitigated by reducing the rate of calls.
    - Spurious account creation: a possible part of the method is automated account creation. This leads to spurious accounts, which imposes a burden on the website's resources. Even though account creation is open and *gratis*, that does not imply there are zero associated costs for the website. Note that previous work has used this exact method [DIP20]: this approach has been deemed acceptable previously (though whether that was justified is unclear).
    - Triggering errors: testing the limits of what the back-end accepts may trigger errors in the back-end (see 'Facebook case study' below for a specific case). In general, developers have intended the back-end API for the designated front-end. They may not have accounted for outsiders just sending data to the back-end outside the front-end – or their front-end might allow data that the back-end cannot handle (e.g., the Facebook case).

3. **Justice.** This research method may expose the lack of security measures a popular website has been implemented, this serves public interest by raising awareness on the topic of password strength and its pitfalls.

4. **Respect for Law and Public Interest.** Most likely this research method will be in conflict with most websites Terms of Service. Ignoring ToS seems fairly typical in scraping-related research.

I would like to ask you for your insights on the ethical aspects of *measuring back-end validation of passwords* as outlined, taking into account the initial ethical review using the Menlo framework. I am happy to provide more information via the email address below, should you have additional questions.

Yours sincerely,

Coen van Driel coenvand@gmail.com Student Computer Science MSc.

Facebook case study For our research proposal we have manually analysed the password requirements of several SSO providers by creating dummy accounts and finding the limits of allowable passwords.

On the Facebook password requirements. Facebook does not actively display password requirements when entering a password. Only when a password fails its validation, the requirements are hinted in the user interface. Facebook does not provide an concrete list of password requirements, but only the instruction that the password needs to be 6 or more characters and 'difficult to guess'. Our tests show that the password 1234567 fails, but 1234567! does pass the validation. Facebook also displays different password requirements on different sign-up pages. Depending on the type of sign-up page, Facebook can also display the requirement 'at least 6 numbers, letters or symbols'. Remarkably, Facebook does not set a limit on password length, long passwords of 100+ characters are successfully submitted, up until the point that the length of the password creates an internal system error and the webpage displays a generic error message.

C. Appendix: Ethical Research Board for Informatics Response

NOTE: This appendix contains the original (Dutch) response from the Ethical Research Board for Informatics (ERBi) from the requested research method evaluation.

Beste Coen,

De ERBi heeft je verzoek met belangstelling gelezen en wenst je allereerst veel succes met dit interessante onderzoek. Hieronder volgt de reactie van de ERBi.

Allereerst, de ERBi had graag een concretere methodologie gezien, maar begrijpt dat je de adviesvraag nu net bewust instuurt op een moment waarop dat advies de methodologie nog kan beinvloeden.

Daarnaast: je brief schetst als context voor je onderzoeksvraag vooral de beveiliging van individuele users. Maar: je mag ervan uitgaan dat een user niet zomaar per ongeluk de front-end omzeilt. Als je hun beveiliging centraal stelt, dan zou het meten van de front-end afdoende moeten zijn. Waarschijnlijk wil je je onderzoeksvraag (en de motivatie in je rapport) dus breder trekken dan alleen de security van individuele gebruikers.

Wat betreft het testen van back-ends, heeft de ERBi de volgende suggesties:

- Let op wat voor websites je bevraagt & in welk tempo. Zelfs in een lijst van topsites komen al snel "kleine" websites voor - websites die niet door een professioneel team van developers 24/7 in de lucht worden gehouden. Populaire self-hosted blogs bijvoorbeeld. Onze suggestie is om uit te zoeken wat breed geaccepteerde rate limiting

waardes zijn en die te gebruiken (en dit in je rapport te documenteren). - Methodologie: om servers niet te snel te overvragen, lijkt het verstandig om niet een "brute-force" aanval te doen, maar gericht te zoeken op de grenzen. Dwz. bijvoorbeeld 1 backend-call met een wachtwoord dat evident gereject zou moeten worden op basis van de front-end checks (bijv: "a"); daarna wachtwoorden testen die net binnen/buiten de grenzen van de client-side checks vallen. Dit in plaats van "a", "aa", "aaa", ... etc. te testen.

- Het probleem met resource-gebruik van aangemaakte accounts zou je, in theorie, kunnen oplossen door de accounts na afloop weer te verwijderen. Waarschijnlijk voert dit veel te ver; onze suggestie is om dit in de overwegingen op te nemen en te expliciteren waarom dit geen redelijk alternatief was. - Bedenk van te voren over hoe je omgaat met responsible disclosure (en, wederom, documenteer dit).

- Tot slot: sommige websites zullen bij account creation serverside een wachtwoord genereren en dat mailen met de verplichting dat te wijzigen bij de 1e keer inloggen. Houdt ook daar rekening mee - niet dat je dit alternatief per se goed af kan handelen, maar dat je er in ieder geval een elegante foutprocedure voor hebt.

Met vriendelijke groet en namens de ERBi nogmaals veel succes met het onderzoek,

Hugo Jonker, Voorzitter ERBi.

Appendix B

Complete manual website analysis results for discovering client-side password requirement indicators

Website	TrancoID	HTML minlength	maxlength	pattern	passwordrules	text	JavaScript text	regex	char.list	algorithm
1. google.com	1						✓			
2. facebook.com	2						✓			
3. microsoft.com	6						✓			
4. twitter.com	7									
5. cloudflare.com	8									
6. netflix.com	10						✓			
7. instagram.com	11									
8. apple.com	12						✓			
9. linkedin.com	13					✓				
10. wikipedia.org	17									
11. amazon.com	19		✓			✓				
12. yahoo.com	25		✓							✓
13. github.com	29				✓					
14. reddit.com	32						✓			
15. pinterest.com	34									
16. zoom.us	41		✓			✓				
17. fastly.net	43					✓	✓			
18. adobe.com	46						✓			
19. vimeo.com	47		✓							
20. openai.com	52									
21. gandi.net	54									
22. wordpress.com	55									
23. bit.ly	61					✓				
24. tiktok.com	65						✓			
25. intuit.com	67		✓							

✓client-side password indicators found; empty: not found

Table B.1: Client-side password requirement indicators on websites

Website	TrancoID	HTML minlength	maxlength	pattern	passwordrules	text	JavaScript text	regex	char.list	algorithm
26. mozilla.org	70			✓		✓				
27. iairgroup.com	75		✓			✓				
28. spotify.com	82			✓			✓			
29. tumblr.com	84									
30. fandom.com	98									
31. nytimes.com	99		✓				✓			
32. ebay.com	100		✓			✓				
33. dropbox.com	101									
34. speechmatics.com	103			✓			✓			
35. imdb.com	104		✓			✓				
36. flickr.com	106							✓		✓
37. ibm.com	116									
38. gravatar.com	118									
39. aliexpress.com	120									
40. soundcloud.com	122	✓	✓							
41. cnn.com	125					✓				
42. webex.com	128						✓			
43. quora.com	139						✓			
44. stackoverflow.com	141					✓				
45. indeed.com	144					✓				
46. theguardian.com	147					✓	✓			
47. cisco.com	149					✓				
48. bbc.com	150					✓				
49. roblox.com	152					✓				
50. sciencedirect.com	156					✓				

✓client-side password indicators found; empty: not found

Table B.2: Client-side password requirement indicators on websites

Website	TrancoID	HTML minlength	maxlength	pattern	passwordrules	text	JavaScript text	regex	char.list	algorithm
51. etsy.com	159									
52. amazon.co.uk	160		✓			✓				
53. twitch.tv	161									
54. booking.com	168						✓			
55. sourceforge.net	170	✓				✓				
56. imgur.com	171		✓							
57. oracle.com	180					✓				
58. researchgate.net	181		✓			✓				
59. dell.com	183		✓			✓				
60. epicgames.com	187		✓			✓				
61. tradingview.com	189					✓				
62. issuu.com	194		✓			✓				
63. freepik.com	195	✓	✓				✓			
64. teamviewer.com	196									
65. weebly.com	199									
66. discord.com	204		✓							
67. hp.com	210		✓			✓				
68. pixiv.net	212									
69. alibaba.com	217									
70. washingtonpost.com	222		✓			✓				
71. linktr.ee	223					✓				
72. reuters.com	227		✓			✓				
73. dailymail.co.uk	229						✓			
74. samsung.com	231		✓				✓			
75. wix.com	233									

✓client-side password indicators found; empty: not found

Table B.3: Client-side password requirement indicators on websites

Website	TrancoID	HTML minlength	maxlength	pattern	passwordrules	text	JavaScript text	regex	char.list	algorithm
76. wiley.com	234					✓				
77. behance.net	238									
78. atlassian.net	240									
79. tinyurl.com	241						✓			
80. bloomberg.com	244		✓			✓				
81. wsj.com	247		✓			✓				
82. godaddy.com	258					✓				
83. ig.com	264		✓				✓			
84. stripe.com	267									
85. ilovepdf.com	268									
86. unsplash.com	277	✓				✓				
87. businessinsider.com	279					✓				
88. hubspot.com	281					✓				
89. ring.com	282	✓				✓				
90. walmart.com	283					✓				
91. wp.com	288					✓				
92. cnbc.com	289									
93. mediafire.com	291		✓			✓				
94. huawei.com	293									
95. unity3d.com	294					✓				
96. slideshare.net	295					✓				
97. kaspersky.com	300					✓				
98. espn.com	307									
99. digitalocean.com	308									
100. hostgator.com	314					✓				

✓client-side password indicators found; empty: not found

Table B.4: Client-side password requirement indicators on websites

Appendix C

Proof of concept configuration

Argument	Reason
–incognito	Avoid cross-website interference
–enable_do_not_track	Avoid cross-website interference
–disable-extensions	Provide a clean environment
–disable-plugins-discovery	Provide a clean environment
–user-agent= Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.0.0 Safari/537.36	Avoid bot detection

Table C.1: Arguments for our Selenium webbrowser automation setup.

Keyword for registration page finding
register
registration
signup
sign-up
createaccount
account/signup
account/create
account/setup
account/sign-up
account/register
account/registration
users/signup
users/register
user/signup
user/register

Table C.2: "URLs with registration keywords" prefixes and suffixes

Appendix D

Complete list of used text patterns

Below we present the complete list of used text patterns for all password requirements..

```
must be (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourteen|fifteen|
at least (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve) chars?
set a password longer than (\d+|one|two|three|four|five|six|seven|eight|nine)
a minimum of (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
(\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? (or more|min)
password must be at least (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
use at least (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
needs to be (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
enter (\d+|one|two|three|four|five|six|seven|eight|nine|ten) or more chars?
length must be (\d+|one|two|three|four|five|six|seven|eight|nine|ten) or more
min length: (\d+|one|two|three|four|five|six|seven|eight|nine|ten)
password min length: (\d+|one|two|three|four|five|six|seven|eight|nine|ten)
password requires (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? min
choose a password at least (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
make it (\d+|one|two|three|four|five|six|seven|eight|nine|ten) or more chars?
ensure (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? min
password must include (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? or more
min password length: (\d+|one|two|three|four|five|six|seven|eight|nine|ten)
password must contain (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? min
use a password (\d+|one|two|three|four|five|six|seven|eight|nine|ten) or more chars? long
must have (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? min
set at least (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? length
(\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? mandatory
at least (\d+|one|two|three|four|five|six|seven|eight|nine|ten) in length
password must be >(\d+|one|two|three|four|five|six|seven|eight|nine|ten)
minimum: (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
```

Figure D.1: Minimum length patterns

```
must be (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourteen|fifteen|sixteen|
must be no more than (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourteen|fi
cannot exceed (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourteen|fifteen|si
at most (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourteen|fifteen|sixteen|
password must be less than (\d+|one|two|three|four|five|six|seven|eight|nine|ten|eleven|twelve|thirteen|fourte
set a password shorter than (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
a maximum of (\d+|one|two|three|four|five|six|seven|eight|nine|ten|twenty|thirty) chars?
up to (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
(\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? or fewer
(\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? max
password length must not exceed (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
password length: max (\d+|one|two|three|four|five|six|seven|eight|nine|ten)
max length: (\d+|one|two|three|four|five|six|seven|eight|nine|ten)
password must contain no more than (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
password should be less than or equal to (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
maximum password length: (\d+|one|two|three|four|five|six|seven|eight|nine|ten)
ensure no more than (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
must not be longer than (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
password requires fewer than (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
cannot be over (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
make sure password is under (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
password must include up to (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
set at most (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
no more than (\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? allowed
(\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars? or less
password must be ≤(\d+|one|two|three|four|five|six|seven|eight|nine|ten) chars?
```

Figure D.2: Maximum length patterns

```
at least one (uppercase|capital)
one or more (uppercase|capital)
requires (uppercase|capital)
use (uppercase|capital)
must include (uppercase|capital)
must have (uppercase|capital)
must contain (uppercase|capital)
needs (uppercase|capital)
password requires (uppercase|capital)
include (uppercase|capital)
```

Figure D.3: Uppercase patterns

```
at least one (lowercase|small)
one or more (lowercase|small)
requires (lowercase|small)
use (lowercase|small)
must include (lowercase|small)
must have (lowercase|small)
must contain (lowercase|small)
needs (lowercase|small)
password requires (lowercase|small)
include (lowercase|small)
```

Figure D.4: Lowercase patterns

```
at least one special
one or more special
requires special
use special
must include special
must have special
must contain special
needs special
password requires special
special characters? required
symbols? like [!@#$%^&*]
non-alphanumeric characters required
must have a symbol
needs a symbol
```

Figure D.5: Symbol patterns

```
contain(?:s)? (\d+) of (?:the )?(\d+) character types?
require(?:s)? (\d+) out of (?:the )?(\d+) character types?
must include (\d+) of (?:the )?(\d+) categories?
at least (\d+) of (?:the )?(\d+) groups?
password must contain (\d+) of (?:the )?(\d+) different character types?
use at least (\d+) of (?:the )?(\d+) types?
```

Figure D.6: Number of required character types patterns

Appendix E

Distribution of length requirements across websites

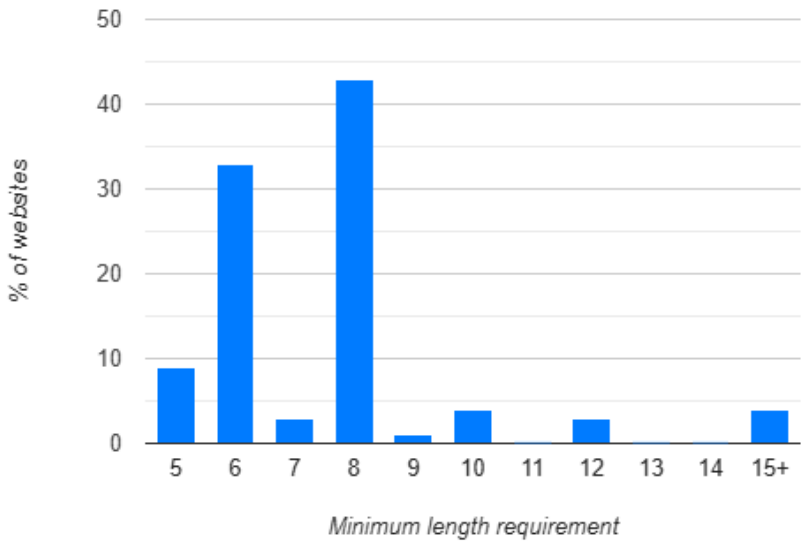


Figure E.1: Distribution of minimum length requirement across tested websites

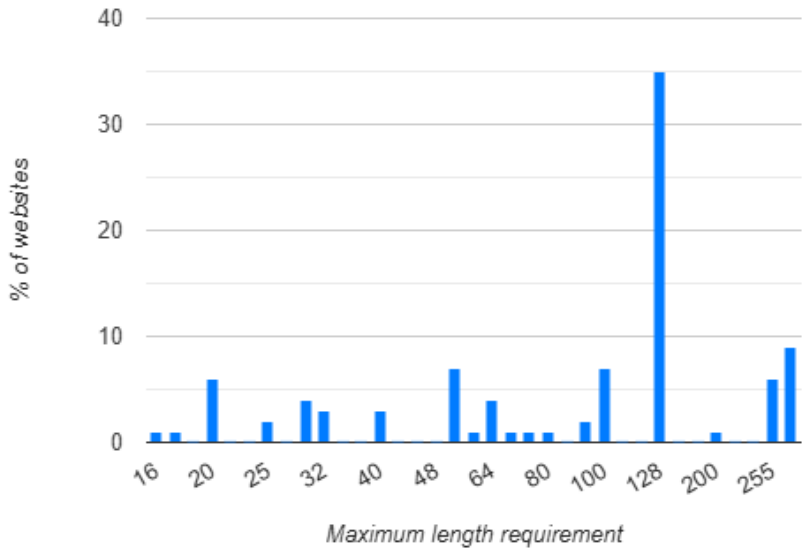


Figure E.2: Distribution of maximum length requirement across tested websites

Appendix F

Case study: university password requirements complete results

University of Applied Sciences	#students	Availability
Hogeschool van Amsterdam	44,600	√
Fontys	44,486	√
Hogeschool Rotterdam	39,000	partial
Hogeschool Utrecht	38,000	√
HAN University of Applied Sciences	37,408	×
Avans Hogeschool	29,900	√
Hanzehogeschool Groningen	29,474	√
Saxion	28,793	×
Windesheim	27,000	√
De Haagse Hogeschool	26,000	×
Hogeschool InHolland	25,000	partial
NHL Stenden Hogeschool	20,000	×
Hogeschool Leiden	14,000	×
Zuyd Hogeschool	13,551	×
Breda University of Applied Sciences	7,223	×
HZ University of Applied Sciences	4,500	×
Amsterdamse Hogeschool voor de kunsten	4,400	×
Hogeschool voor de Kunsten Utrecht	4,400	×
Van Hall Larenstein	4,293	×
Christelijke Hogeschool Ede	4,200	×
HAS green academy	3,400	×
ArtEZ University of the Arts	3,000	√
Aeres Hogeschool	2,500	×
Hogeschool Viaa	2,113	×
EMarnix Academie	1,800	×
Hogeschool der Kunsten Den Haag	1,686	×
Driestar hogeschool	1,571	×
Hogeschool IPABO	1,212	×
Hogeschool KPZ	1,075	×
De Kempel	1,043	√
Codarts Rotterdam	1,000	√
Thomas More Hogeschool	1,000	×
Gerrit Rietveld Academie	850	×
Design Academy Eindhoven	700	×
Inselinge Hogeschool	400	×

Table F.1: Public availability of password policies of universties of applied sciences.

Appendix G

Survey: Password policies in organisations: Password requirements and password management

Password policies in organisations

Special Note: Despite this survey being presented in English, all answers may be answered in English or Dutch.

Thank you for participating in this short survey on password policies in organisations!
This survey is aimed at professionals who are involved in the cybersecurity/information security space within an organisation. If you are not involved in this space, please forward this survey to a colleague that is.

This survey is grouped into 3 parts, the estimated time to complete this survey is 5-10 minutes.

About this survey
This survey is part of the Masters thesis Computer Science at the Open Universiteit by Coen van Driel. This research is supervised by assistant professor Dr. Ir. Hugo Jonker.

How the data from this survey is used regarding your privacy
The results of this survey are anonymised. When asked for your organisation's name, this is only for communication purposes in case we want to send participation reminders. This is not used in our result analysis.

The results of all surveys are aggregated and analysed in our study.
We will not link or mention any particular organisation by identifiable properties such as organisational name.

Contact
If you have any questions regarding this survey, please reach out via: coenvand@gmail.com

Academic institution details
Open University of the Netherlands (Open Universiteit)
Valkenburgerweg 177, 6419 AT Heerlen, Netherlands
https://www.ou.nl/

There are 12 questions in this survey.

Before we start

1
Please enter the survey code you were provided.

This information is only used to keep track of which invitee has yet to complete the survey.

Please write your answer here:

Part 1 of 3: About your organisation

In this section we ask about the size and cybersecurity team of your organisation to add additional context to your answers.

2
What is your role within your organisation?

Select all that apply
Please choose **all** that apply:

- [ ] IT manager
- [ ] CISO
- [ ] (Information) Security officer
- [ ] Compliance/privacy/risk officer
- [ ] IT auditor
- [ ] Security architect
- [ ] Security engineer
- [ ] Security analyst
- [ ] Other:

3
What *roughly* is the size of your organisation?

This refers to the number of people of your entire organisation (including both parttime and fulltime), not a single organisational unit or team.

Please write your answer here:

4
How many people *roughly* are involved in cybersecurity/information security within your organisation?

Please elaborate on the organisational roles of those involved.

E.g. 1x CISO, 1x IT manager, 3x Security/compliance officer...

Please write your answer here:

Part 2 of 3: Your organisation's password policy

In this section we ask about the password policy of your organisation.

What is a password policy?
A password policy is a policy that sets specific rules for how to create and manage passwords. This policy can be composed of both password composition rules (e.g. a minimum length for passwords) as well as password management rules (e.g. it is required to use a password manager).

An organisation may have a single policy, or multiple policies depending on the use case (e.g. Single Sign-On password requirements may differ from Third Party Software password requirements).

In this survey we are interested in the **main or default** password policy that is used for most use cases within your organisation. This is often a Single Sign-On enforced policy.

5

For your organisation's primary software applications/systems, can users create their own passwords?

E.g. Yes, when creating new accounts or when resetting passwords, users can create their own password through the Single Sign-On provider...

E.g. No, users get auto-generated passwords via email with no option to customize their own passwords...

Please write your answer here:

6

If you have answered 'No' to the previous question, please skip this question.

Which of the following requirements are included in your organisation's password composition policy for creating new passwords?

Please check all requirements that apply and, when applicable, use the textbox for the requirement value.

Use the 'Other'-field for additional requirements that are explicitly included or excluded in your organisation's policy.

Comment only when you choose an answer.

Please choose all that apply and provide a comment:

☐ Minimum length (e.g. 15)

☐ Maximum length (e.g. 128)

☐ Must include specific charactertypes (e.g. alphabetic, capitalized, numbers, symbols)

☐ Allow spaces

☐ Reject sequential characters (e.g. 123)

☐ Reject known bad/leaked passwords (e.g. HaveIBeenPwned.com leaked password database)

☐ Allow non-ASCII characters (non-latin symbols)

Other:

7

To what extent is this password composition policy enforced within your organisation?

E.g. this policy is technically enforced on Single Sign-On passwords but not on non-SSO software...

Please write your answer here:

8

Does your password policy include a password expiration policy?

E.g. passwords expire and must be re-created every 90 days...

E.g. passwords never expire...

Please write your answer here:

9

In what way does your organisation require Multi-factor authentication (MFA) for applications/systems?

E.g. only for internal systems MFA is required...

E.g. only for mission critical systems MFA is required...

Please write your answer here:

10

In what way does staff recover passwords when forgotten?

E.g. using a secret question and answer to verify identity...

E.g. a self service portal that sends a reset-link to their email...

Please write your answer here:

Part 3 of 3: The reasoning behind your organisation's password policy

In this section we ask about the reasoning behind your organisation's password policy.

11
What is the basis for your organisation's password management policy?

Please elaborate.

E.g.

- *Specified by cybersecurity standard (e.g. ISO27001, NEN7510)*
- *Researched by staff.*
- *Recommended by cybersecurity consultant.*
- *Based on the NIST Password Guidelines.*
- *Based on the Microsoft password guidelines.*
- *Default configuration by software (e.g. Office365 baseline).*
- *To align with password requirements by third party vendors/applications.*

Or a combination of these examples.

Please write your answer here:

12
What is the policy for evaluating and/or updating the password management policy?

E.g. the policy is evaluated and updated each year as part of an internal audit cycle...

E.g. only when a staff member decides to do so...

Please write your answer here:

You have completed the survey, thank you!

Submit your survey.

Thank you for completing this survey.