Politecnico di Torino

Master Degree Course in Computer Engineering
Academic Year 2022/2023
October 2023

Usability of Safety Critical Applications in Enterprise Environments

Defining Guidelines for Error Preventing UI/UX Patterns and Improving Existing Interfaces.

Supervisor:
Fulvio CORNO

Candidate:
Gabriele SAMBIN

Table of Contents

Chapter 1: Introduction

1.1 Motivation and context of the thesis

The present thesis has the objective to investigate how different design choices can influence user behaviour in the context of cybersecurity, with a focus on enterprise environments. Starting from a review of existing literature, different solutions are analysed, to then define the guiding concepts of human computer interaction and ways to apply them in the chosen scope. The thesis uses the work done during an internship at aizoOn on a network monitoring application called IDS (fictitious name) to furtherly explore options, tools and methodologies developer have to protect sensible assets starting from the interface.

Considering that the application taken in exam recently underwent a modernization process, a series of guidelines were produced, with the objective of improving usability of a safety critical application. Said guidelines were used to recognize old version's issues and aid in the updating of the rearchitected version's interface.

Success of the guidelines was determined based on the impact they had on the application and the action taken by aizoOn as a response to their content.

This document finds its value in highlighting the importance of understandability of information to take correct decisions: IDS is an application powered by artificial intelligence that provides network risk monitoring to analysts, so taking an action without properly understanding the program's output can potentially be catastrophic. Conveying the right meaning requires presenting data in a way that cannot be misinterpreted, with complete accessibility options and without frustrating patterns that might distract users, all while assuring that the security of the application is immediately perceivable.

The choice of developing guidelines also depended on the need to bring together separate sets of general best practices in one consultable document, personalized to fit the need of the particular case of IDS.

1.2 Importance of web application security

The most important parts of an information are its understandability and its reliability, obviously an information that cannot be understood cannot be used and later in the thesis there will be an extensive exploration of how to guarantee that, but if an information is not reliable any action taken in response to acquiring it will be potentially dangerous. Security of an information as described by the confidentiality, integrity and availability triad is imperative for any communication (with the appropriate forms and limits). Web applications are, right now and possibly even more in the future, the main vehicle of information so they should be the main focus of information security.

Furthermore, the businesses related to the web keeps growing in size and number: global application development market grew from $260.79 billion to $334.86 billion from 2022 to 2023, at a CAGR of 28%[1] and is expected to keep growing, both making the attacks against them more worth and making it easier to find targets with exploitable weaknesses. Protecting one single application is already a challenge, guaranteeing security for all of them is nearly impossible, for this reason it's important to continuously seek new ways to achieve security.

1.3 Web application security: definition

Before starting to analyse how to secure a web application it's needed to define what web application security is, putting together a couple of commonly accepted definitions:

- "Web application security is the practice of protecting websites, applications, and APIs from attacks. It is a broad discipline, but its ultimate aims are keeping web applications functioning smoothly and protecting business from cyber vandalism, data theft, unethical competition, and other negative consequences." [2]

This definition by Cloudflare focuses on security as a practice (and, as such, something that has to be done over time) and on what the absence of security involves.

A 2003 article from Microsoft[3] gives a similar definition, again including the threats themselves as a part of it.

- "Security is fundamentally about protecting assets. Assets may be tangible items, such as a Web page or your customer database - or they may be less tangible, such as your company's reputation. Security is a path, not a destination. As you analyse your infrastructure and applications, you identify potential threats and understand that each threat presents a degree of risk. Security is about risk management and implementing effective countermeasures."

Both definitions agree on security being a process and in particular one shaped by the possible menaces for the assets to be protected, web application security consists in applying said process to web applications.

Possible threats continuously evolve and as the codebase of an application expands new vulnerabilities will be generated and eventually discovered, securing a system means constantly updating in by recognizing weaknesses before attackers or incidents exploit them.

1.4 Web application vulnerabilities

To understand how to secure a web application is needed to understand what it should be protected from. The OWASP Foundation publishes a top 10, updated in 2021, of the most relevant types of vulnerabilities in web applications, with statistics about each one[4]:

#	Name	CWEs Mapped	Max Incidence Rate	Average Incidence Rate	Average Weighted Exploit	Average Weighted Impact	Max Coverage	Average Coverage	Total Occurrences	Total CVEs
1	Broken access control	34	55.97%	3.81%	6.92	5.93	94.55%	47.72%	318,487	19,013
2	Cryptographic failures	29	46.44%	4.49%	7.29	6.81	79.33%	34.85%	233,788	3,075
3	Injections	33	19.09%	3.37%	7.25	7.15	94.04%	47.90%	274,228	32,078
4	Insecure design	40	24.19%	3.00%	6.46	6.78	77.25%	42.51%	262,407	2,691
5	Security misconfiguration	20	19.84%	4.51%	8.12	6.56	89.58%	44.84%	208,387	789
6	Vulnerable and outdated components	3	27.96%	8.77%	5.00	5.00	51.78%	22.47%	30,457	0
7	Identification and authentication failures	22	14.84%	2.55%	7.40	6.50	79.51%	45.72%	132,195	3,897
8	Software and data integrity failures	10	16.67%	2.05%	6.94	7.94	75.04%	45.35%	47,972	1,152
9	Security logging and monitoring failures	4	19.23%	6.51%	6.87	4.99	53.67%	39.97%	53,615	242
10	Server-side forgery request	1	2.72%	2.72%	8.28	6.72	67.72%	67.72%	9,503	385

OWASP's top 10 is a good starting point to identify which fields need to be operated on to perfect security.

The insertion of "Insecure design" in the list, especially in such a high place, should be a relevant enough alarm bell for designers and developers, because it highlights how an important number of security issues depend on conceptual errors.

Design errors in user interfaces directly interact with the other hard-to-encompass factor in security, which is human error; better UI/UX design can help prevent it and the measure of this impact is among the key findings of this thesis.

1.5 Trends and statistics on web application attacks

Web applications are an ever-changing environment and as such understanding how this change is happening is necessary to take any kind of action. Several new trends have sprung in the last years, some of them as a consequence of the Covid-19 pandemic. This chapters analyses some of the most discussed trends to gain an insight on how the environment is changing and what future challenges might emerge.

S. Moore in an article for Gartner summarizes some relevant statistics:[5]

1. Attack surface expansion: 60% of knowledge workers are remote, and at least 18% will not return to the office. These changes in the way we work, together with greater use of public cloud, highly connected supply chains and use of cyber-physical systems have exposed new and challenging attack "surfaces."

As of 2023, 37% of knowledge workers still work hybrid (and 14% are fully remote), despite the end of the pandemic; a lot of these workers aren't used to work with this kind of mechanisms, furtherly weakening the defences of the increased surface.[6]

2. Identity system defence: Misuse of credentials is now a primary method that attackers use to access systems and achieve their goals. For example, in the SolarWinds breach attackers used a supplier's privileged access to infiltrate the target network.

The danger this poses is great because it completely nullifies the safety provided by the algorithms, furthermore it's a human error that does not depend, at least directly, from the application's design, but only from users' knowledge, making it harder to limit.

3. Digital supply chain risk: Gartner predicts that by 2025, 45% of organizations worldwide will have experienced attacks on their software supply chains, a three-fold increase from 2021.

Attacking an organization's software supply chain can grant the attacker access to the organization's systems or just impair their functionalities, giving an unfair advantage to possible competitors. When developing a safety critical application every vulnerability will expose both the user's organization and the developer's organization. This statistic also means that dependencies used in development can be responsible for a relevant number of these vulnerabilities.

4. Vendor consolidation: Security products are converging. While it may introduce new challenges such as reduced negotiating power and potential

single points of failure, Gartner sees consolidation as a welcome trend that should reduce complexity, cut costs and improve efficiency, leading to better overall security.

75% of organizations are relying on a smaller number of vendors [7], this means that the products offered will be subject to more testing and subsequently provide a more secure and efficient service, still it will be progressively harder for businesses to insert themselves in the market and that could limit innovation.

5. Cybersecurity mesh: a modern conceptual approach to security architecture that enables the distributed enterprise to deploy and integrate security to assets, whether they're on premises, in data centres or in the cloud.

Cybersecurity mesh is an architecture based on interoperability and coordination between different security tools. The integration of coexistent products improves the capillarity of centralized security operations.

7. Beyond awareness: Human error continues to feature in most data breaches, showing that traditional approaches to security awareness training are ineffective. Progressive organizations are moving beyond outdated compliance-based awareness campaigns and investing in holistic behaviour and culture change programs designed to provoke more secure ways of working.

Human error prevention fits into the scope of this thesis, yet the traditional strategy of focusing on user/developer education has proven itself to be not sufficient. The main reason for this is that conscious actions require concentration and energy, that a user focused on productivity just doesn't plan to invest; even for everyday use people care more about their tasks being fast and simple and only consider security-oriented actions if they don't limit these characteristics. Security designers should strive to create digital environments in which users are led to the most secure option by default, either with explicit suggestion or with implicit guidance. Forcing the users into "healthy" patterns is mandatory to effectively reduce human error.

According to recent research from Verizon[8] "More than 30% of breaches-causing errors are misconfiguration errors of web applications"

In an enterprise setting the software used for this purpose might be part of the problem: if decisions about a system's configuration are taken based on data provided by an application the way it presents data to its users can condition those decision to the point of changing their outcome.

From Terranova's 2021 Gone Phishing Tournament report we can notice that: "Nearly 1 in 5 recipients clicked on the phishing email link included in the simulation's initial message." And "In addition, over 14% of all end users who encountered the scenario ultimately failed to identify the simulation's webpage as unsafe and clicked on a link to download a malware file. Globally, more than 70%

of the phishing simulation’s clickers went on to obtain the file from the phishing webpage.”

This shows the need for legitimate organizations to be easily recognizable, because phishing or equivalent mechanisms can also be performed through a web page, a factor that highlights the importance of a security-conscious interface in application for the public and for the enterprise setting.

A supplementary chapter of the produced guidelines focuses on preventing brand impersonation and limiting phishing practices through user interface, as, again, user education is simply not efficient in limiting this kind of attacks.

Chapter 2: Securing Legacy Applications to Meet Modern Security Standards

2.1 Overview of legacy applications and their security challenges

In the ever-expanding and changing world of web applications it's normal for new applications to appear and for old one to get discontinued. If a piece of code is meant for the end user interrupting support generally implies making it not accessible anymore, but when it's a necessary component or it's a dependency of other applications this is just not possible. Furthermore, a lot of enterprises still use legacy systems on purpose for a variety of reasons, like the impossibility of interrupting a service, the complexity of the migration or the cost of the system's modernization. Using legacy systems has different consequences: [9]

- More expensive maintenance: older applications might have been conceived for a "slow" changing setting, have poor documentation and sometimes the know-how needed to maintain them is not present in the developer-base.
- Performance: old technologies may not be suitable for modern environments and may have difficulties in handling the high amount of data.
- Compliance: regulations exist to protect users, but some legacy systems could not have the capability to comply, causing owners to incur in fines.
- Security: some software not receiving updates means that any vulnerability that will be found will never be patched, infinitely extending the window of exposure; obviously this also becomes a problem for any applications that use it.

Comparing Owasp's statistics[4] on vulnerabilities one can notice that Vulnerable and outdated components have the highest average incidence (8.77%), yet it's the one with lowest testing coverage (51.78% at best). Legacy applications, even when updated, tend to keep the same dependencies, so finding ways to secure old components is a priority.

In general, leaving open vulnerabilities in the dependencies is incredibly dangerous and has been responsible for the spread of different attacks through the years.

IDS, after having its front-end re-platformed from Angular to React, still uses some of the older dependencies, identifying which of them are not supported anymore and need to be changed or adapted is a mandatory first step in making the application secure and usable.

2.2 Strategies for updating legacy applications to meet modern security standards

To correctly identify strategies for updating legacy application its first needed to define what exactly is this kind of update. Using a definition of application modernization from industry analyst David Weldon:[10]

“Application modernization is the process of taking old applications and the platforms they run on and making them ‘new’ again by replacing or updating each with modern features and capabilities that better align with current business needs.”

There can be a lot of different ways to update an application, covering different scopes and objectives, thankfully Gartner defines 7 application modernization approaches[11]:

- Encapsulate: reusing legacy components in a new architecture, extending their functions. This is done by accessing via API the encapsulated components. Encapsulating is low risk, but requires attentive analysis on how the legacy software’s vulnerabilities could be exploited by means of those APIs. It’s the main way to deal with legacy libraries.
- Rehost: redeploying components without modifications to another infrastructure. This method is quick but implies leaving vulnerabilities exposed.
- Replatform: migrating code to a new platform, changing the code, but keeping structure, features or functions. It can be limited by the external dependencies used by the project not being compatible with the new platform.
- Refactor: refactoring the code in order to optimize it and to remove issues depending on its outdated parts, keeping the same functionalities. Refactoring can be time consuming, especially for large applications, but it’s a good compromise.
- Rearchitect: altering the application into a new architecture.
- Rebuild: completely rewriting the application, but keeping the same scope.
- Replace: retiring the old application and defining a new one to be developed.

Choosing the correct one among these seven approaches, ordered by increasing scope (and cost as a consequence), is mandatory to perform a correct modernization.

2.3 Best practices for securing legacy applications

The first step to secure any application is to analyse it and model (and rank) potential threats, this is also true for legacy applications: understanding how they interact with external entities and determine how those interactions can expose the system is essential to the process. Once the threats have been identified one can select the proper countermeasures.

2.3.1 Access limitation

If an application is just too old to be secure it's useful to isolate it as much as possible, this can be done on different levels:

- Implementing multi factor authentication limits access on a user level.
- If the system needs to be connected to the internet using a VPN can control network access.
- Code access can be reduced through micro-segmentation, separating components and only using what is needed, this can also make it easier to maintain the application, as it reduces the compatibility issues that may arise between already updated and non-updateable libraries of the application.

2.3.2 Code modification

If, when securing a legacy application, one needs to modify the codebase, the structure of the process must consider exclusivity of entities as the most important factor to determine priority: starting from exclusive components means decreasing interdependencies and improving workflow.

Furthermore, it's important to remove dead code during modernization, even if it doesn't impact program's behaviour, because it will increase maintenance cost and could expose exploitable vulnerabilities.

2.3.3 Dependencies

Legacy applications use legacy external libraries, so those libraries need to be secured themselves. Often older libraries are not supported anymore by the original authors or are just not fully compatible with the new environment, in these cases developers need to find some kind of workaround: creating custom version of the libraries is good to keep the same structure previously used, but increases the testing surface and requires constant updating, on the other hand switching to new components requires a higher cost upfront in terms of time, as the existing code will need to be changed, but is safer and cheaper on the long run, because the library will be updated outside of developer's responsibility.

To better verify the state of IDS security a vulnerability scan was performed and three dependency vulnerabilities were found, but it's important to notice that the vulnerable packages were used by a total of seventeen instances of direct dependencies, greatly increasing the attack surface. All vulnerabilities found are to regular expression denial of service, to exploit CWE-1333[12] attackers could force the system to consume excessive resources, by providing an input able to put the "backtracking" feature in a sort of loop.[13]

Two of them, one found in the "semver" package[14], a semantic version parser, and one found in the "word-wrap" package[15], used to wrap words to a specified size, were easily solvable, as there was a patched version available that "npm audit -fix" was able to find. However, the "d3-color" vulnerability[16] wasn't so trivial to solve: the module used for handling colour spaces by different dependencies of ant-design components, but "@ant-design/charts" indirectly depends on the non-patched version. The chosen approach was to replace the library altogether, as the authors are just not supporting it anymore, so relying on a workaround would just postpone the problem to the next discovery of a vulnerability. In Appendix A the full report of command outputs can be found.

This brief experiment was successful in showing how, even if the code base of an application is secure the external components used can become extremely dangerous when left unchecked.

2.4 Modernization tools

A variety of tools exists to help developers put code migration in practice, especially with the recent increase in the use of artificial intelligence in development environments.

Grit[17] for example is a software maintenance tool, that uses AI to automatically modernize code, even translating Angular to React through a query language called GritQL.

Tools like that can be a great starting point for the modernization of a piece of software, but for a complete and fulfilling migration supporting them with actual know-how is mandatory.

In addition using static analysis tools (like SonarQube[18]) is strongly suggested to try and optimize the transition, even if they require more human intervention.

It must be noted that a big part of the work needed to modernize an application's front-end stands in adapting the user interface, yet there is a general absence of tools to help with this process and it would clearly be incredibly complicated to even conceive one. IDS's transition was mainly done without dedicated supporting tools and as a consequence a number of issues are still being resolved.

Chapter 3: Best Practices and Patterns for Front-End Development and UI/UX for Usable Safety Critical Applications

3.1 Importance of front-end development in web application security

As previously stated, before assessing the usability of a safety critical application it is needed to verify the security of it: the present chapter is meant to present some commonly recognized existing guidelines and patterns to guarantee this security.

How can easily be extracted from the Owasp Top 10 a lot of web applications' vulnerabilities can either be in the back-end, but accessible through front-end exploits or directly located in the front-end. In particular:

- Broken access control could depend on front-end receiving (and exposing) too much information from the back-end or giving access to too much privilege.
- Injections are clearly dependent on the front end, as they can be prevented by validating and sanitizing user inputs.
- Insecure design and Security misconfiguration can affect every part of the code, putting an unnecessary feature in a web page might lead to dangerous behaviours.
- Vulnerable and outdated components are especially present in the front-end: web applications tend to use a lot of external components and, mainly in legacy applications, they might have known vulnerabilities that are not patched (or are not patchable at all because they're not supported anymore).
- Identification and authentication failures can depend on front-end errors, like dangerous default settings or excessive reuse of session ids.
- Software and data integrity failures is a case in which the front-end can be damaged by data coming from the back-end (real or presumed). Not implementing integrity check or proper serialization can lead to presenting the user some manipulated data. A similar situation is seen with Cryptographic failures.
- Security logging and monitoring failures is another sensible topic as logs are often kept locally to optimize data transfer efficiency.
- Server-side forgery request can also be limited with appropriate input validation.

Agreeing on accepting a well-tested set of best practices for front-end development is a mandatory step in order to propose usability guidelines that are meant to be used by developers, as said guidelines must not contradict or limit the security of the code itself.

3.2 Best practices for front-end development

3.2.1 Security oriented

When developing a front-end architecture following existing guidelines and best practices is the best way to prevent common errors and guarantee the absence of the most trivial vulnerabilities. An article from CLIMB presents a top 10 of relevant best practices:[19]

1. Use HTTPS: against man-in-the-middle attacks, Google now considers it a SEO ranking factor.
2. Avoid Inline Code and Eval (): inline code is difficult to maintain and debug. Eval () is a function in JavaScript which evaluates a string as if it were an expression.
3. Don't Trust User Inputs
4. Sanitize Outputs: against cross-site scripting (XSS) attacks
5. Implement a Content Security Policy (CSP): defines which sources are allowed to load content on your website
6. Set HTTP Headers: can be used to control how the browser behaves when it receives content from the server (can force https): primarily used as a HTTP response header, you can also apply it via a meta tag
7. Keep Your Software Up-to-Date
8. Protect Sensitive Data with Encryption
9. Securely Store Passwords
10. Prevent Clickjacking Attacks: frame-busting techniques such as X-Frame-Options or Content Security Policy (CSP).

3.2.2 Scalability oriented

On the other hand, Outsystems defines rules to build scalable front-end architectures:[20]

- Define the HTML Layout structure without style rules, as a skeleton to start working from.
- Build without business logic. Use functional logic only.
- Avoid using animations in JavaScript. Build these in CSS, so it can be easily overridden. By doing this the work of processing animations is done in the Graphics Processor Unit instead of the CPU.

- Avoid manipulating the attribute style in elements. Add classes instead, and document it. With this approach, you are not forcing things that people may not want in a particular scenario, and you avoid cleaning all the attributes by mistake.
- Avoid polluting your structure with elements just to cover a specific case. Instead, use pseudo-elements. With this approach, you keep the same structure applied in different themes, and some of them will use pseudo-elements to cover specific cases.
- Control your CSS base.
- Use Patterns.
- Define your own rules for scalability.
- Create a platform to centralize all information.

3.2.3 Fluidity oriented

PixelMatrix instead suggest a list of five best practices related to offering a more fluid and fast experience[21]:

1. Subside Resources: eliminating all unwanted data or characters from your HTML, CSS, JavaScript, or jQuery code
2. Limit or Minimize the Number of Server Calls: Typically, the more calls your frontend makes to the server, the longer it takes to load.

More server calls mean more traffic analysable by potential attacker

3. Eliminate Unnecessary Custom Fonts: huge performance cost

If multiple fonts are used, they can create problem distinguishing legitimate content from ads

4. Minimize Files: The larger the files, the more time it takes to render.
5. Try Lazy Loading: improves front-end loading time

Users should have a way to distinguish pop-ups from regular content. Lazy loading should not involve page structure, just page content.

3.3 Strategies for preventing or mitigating front-end security vulnerabilities

Some strategies are suggested by OWASP as solutions to the top 10 types of vulnerabilities[4]:

1. Broken access control:
    - Access control should be kept out of front-end, when possible, an application client could be modified by the user.
    - Denying access by default to anything that’s not needed (without even send data for pages that should be inaccessible)
    - Re-using the same mechanism for the whole application, minimizing CORS usage
    - Enforcing record ownership
    - Logging access control failures and alerting admins when appropriate (e.g., repeated failures).
    - Developers and QA staff should include functional access control unit and integration tests.
2. Cryptographic failures:
    - Correctly identifying which data is sensitive
    - Not storing unnecessary data
    - Encrypting all data at rest
    - Ensuring up-to-date and strong standard algorithms, protocols, and keys are in place; using proper key management.
    - Encrypting all data in transit with secure protocols such as TLS with forward secrecy (FS) ciphers, cipher prioritization by the server, and secure parameters. Enforcing encryption using directives like HTTP Strict Transport Security (HSTS).
    - Disabling caching for responses that contain sensitive data.
    - Applying required security controls as per the data classification.
    - Not using legacy protocols such as FTP and SMTP for transporting sensitive data.
    - Always using authenticated encryption instead of just encryption.
    - Verifying independently the effectiveness of configuration and settings.
3. Injection:
    - Keeping data separate from commands and queries
    - Implementing controls and limiting inputs also in the client
4. Insecure design:
    - Establishing a secure development lifecycle with AppSec professionals to help evaluate and design security and privacy-related controls
    - Using a library of secure design patterns or ready to use components
    - Using threat modelling for critical authentication, access control, business logic, and key flows
    - Integrating security language and controls into user stories
    - Integrating plausibility checks at each tier of your application (from frontend to backend)

   - Writing unit and integration tests to validate that all critical flows are resistant to the threat model. Compile use-cases and misuse-cases for each tier of your application.
   - Segregating tier layers on the system and network layers depending on the exposure and protection needs
   - Segregating tenants robustly by design throughout all tiers
   - Limiting resource consumption by user or service
5. Security misconfiguration:
   - Automating setup for new secure environments
   - Platform should be minimal and without unnecessary features
   - Configuration update should be included in patch management process
   - Implementing effective and secure separation between components
   - Sending security directives to clients
   - Automating verification of configurations' effectiveness
6. Vulnerable and outdated components:
   - Removing unused dependencies, unnecessary features, components, files, and documentation.
   - Inventorying the version of components and dependencies and monitoring their vulnerabilities.
   - Only obtaining components from trusted sources
   - Monitoring for libraries and components that are unmaintained or do not create security patches for older versions.
   - Using virtual patches
7. Identification and Authentication Failures:
   - Implementing multi factor authentication
   - Never deploying with default credentials
   - Checking for weak passwords
   - Aligning password length, complexity, and rotation policies with National Institute of Standards and Technology (NIST)
   - Ensuring registration, credential recovery, and API pathways are hardened against account enumeration attacks by using the same messages for all outcomes.
   - Limiting or increasingly delaying failed login attempts, but be careful not to create a denial-of-service scenario. Log all failures and alert administrators when credential stuffing, brute force, or other attacks are detected.
8. Software and Data Integrity Failures:
   - Using digital signatures to verify software and data.
   - Using libraries and dependencies from trusted repositories.
   - Using a software supply chain security tool.
   - Reviewing process for code and configuration changes.
   - Verifying CI/CD pipeline has proper segregation, configuration, and access control

   - Performing integrity checks on unsigned or unencrypted data
9. Security Logging and Monitoring Failures
   - Ensuring all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.
   - Ensuring that logs are generated in a format that log management solutions can easily consume.
   - Ensuring log data is encoded correctly to prevent injections or attacks on the logging or monitoring systems.
   - Ensuring high-value transactions have an audit trail with integrity controls to prevent tampering or deletion, such as append-only database tables or similar.
   - DevSecOps teams should establish effective monitoring and alerting such that suspicious activities are detected and responded to quickly.
   - Establishing or adopting an incident response and recovery plan, such as National Institute of Standards and Technology (NIST) 800-61r2 or later.
10. Server-Side Request Forgery:
    - For frontends with dedicated and manageable user groups using network encryption on independent systems to consider very high protection needs

3.4 User Interface Rules and Patterns

3.4.1 8 Golden Rules

The 8 Golden Rules of Interface Design[22] defined by Ben Shneiderman are a set of general usability rules for user interfaces, their main objective is to help provide a positive experience, but in the context of a safety critical application contravening them could create in the UI what is functionally a vulnerability:

1. Strive for consistency

Inconsistencies can cause problems for security, because of misinterpretation of the effect of specific actions. Users may also recognize a weirdly looking component as an external and possibly dangerous item, reducing the trust in the application.

2. Cater to universal usability

Users with a different skillset than expected might not be able to use the system properly (limited language options, no support for visually impaired people), this can generate the same problems mentioned in the previous point.

3. Offer informative feedback

No feedback might be confusing and lead user to repeat an action, not knowing if what they tried to do was successful, wrong or misunderstandable feedback might be misleading, too much feedback might be irritating and needlessly slow the execution of tasks.

4. Design dialogs to yield closure

Not having a clear beginning and end of tasks may convince users that they're still performing a completed task, possibly giving information that they're not meant to give. An attacker could create an ad that looks like a piece of a form, having inattentive users click on it. Also, a clear end of a task gives users a sense of satisfaction that can improve they experience (and productivity).

5. Prevent errors

Options that are not applicable or would create an error should be disabled preventively, this includes illegal inputs. Furthermore, offering clear and simple recovery instructions can prevent additional errors due to users trying to fix things the wrong way. Errors should not alter application state or there should be at least to recover the old state.

6. Permit easy reversal of actions

Similarly, to the previous point, there should always be an "easy" way to revert an action and if there's not the user should be properly warned and asked for confirmation.

7. Keep users in control

Tedious tasks will make the user skip through it without paying too much attention. If the interface is not responding the user might keep trying to do things although it's not the correct way or do other random things hoping for a reaction. Impairing the sense of agency of a user might make them uncomfortable and generate some oppositional behaviours.

8. Reduce short-term memory load

Asking users to remember too much info, especially from one view to another, is uselessly taxing on their concentration. People not remembering what they put in an earlier input may lead to inconsistencies in information, which can obviously create other problems if not handled, or to a useless loss of time.

3.4.2 Intentionally deceptive designs and unintentional use of them

A malicious way to design interfaces is constituted by "intentionally deceptive designs"[23] used by some companies to force users into doing actions against their best interest. While obviously an honest company would never want to implement them, it's possible to accidentally reproduce some of them or to indulge in bad practices similar enough to have the same effects.

- Confirm-shaming: it consists of triggering negative emotions to influence users' decisions; this can happen because of bad wording in confirmation pages that makes one option look worse than the others. As discussed previously leading users to particular choices can be dangerous.
- Nagging: it's a form adversarial resource depletion, in which user are constantly interrupted and asked to perform a determined action, until they accept to do it. Every application has some sort of notification, a careless use of them can be perceived as nagging and asking the same thing multiple time can lead users to change their answer out of exhaustion hoping to stop the interruptions. For employees the need to "just do their job" will speed up this process.
- Obstruction: the interface makes it difficult to complete a task by creating artificial limitations. Even in legitimate applications it's easy to find some unnecessarily complicated input, sometimes it's because it may look cool, sometimes it's because not enough effort has been put to make it easier, in both cases users are inconvenienced.
- Preselection: users are presented with a default option that has already been selected. This is another way to influence a decision and, again, is just as dangerous as the others. In a good interface the default option should be the safest one or, if possible, non-selectable.
- Sneaking: relevant information is hidden or delayed so that users don't understand the implications of their actions. It's possible to implement this on accident by giving warnings too late or putting information in places only accessible after taking compromising actions.
- Trick wording: when presented with a lot of information people tend to scan-read instead of over-analysing every word, ambiguous language can exploit this to confuse users. If the proper effort isn't put into the language that is being used it's easy to inadvertently apply this pattern.
- Visual interference: disguising information by positioning it in weird places or making it physically hard to see. Web designer should consider user behaviours like banner blindness[24] and contrast guidelines in order to make all the content clearly visible.

Leading users into situations they don't want to be in can be considered a self-inflicted threat to the system and has complex security implications, especially in an enterprise setting and even more if it's an application that provides some security related services.

Chapter 4: Design Principles for Establishing New Guidelines

When defining any practical rule to apply to a system the efficacy of it strictly depends on the quality of the model it's based on and in human-computer interaction any model used is inherently a model of human's cognitive abilities. This chapter defines principles and concepts needed to create a model of user's behaviours, from which guidelines will be extracted.

An article[25] from year 2000 already complains about regular computer security principles (like random and frequently changed passwords) becoming useless if we consider the humans that will use said passwords, yet more than twenty years later no real action has been taken in that direction apart from a few cases, closer to experiments than to globally diffused practices, and even when those opportunities are offered by tools they are just not enforced by organizations. It's important to advocate for more secure designs, that do not rely on users' knowledge or effort as "It's simply unrealistic to assume that average users can keep up with them [the attackers]. The only real solution is to make security a built-in feature of all computing elements."[26]

However, it's just unfair to blame users, when some designer's choices set them up to fail[27], also stressing them too much about potential threats can generate feelings of uncomfortableness and make them loose trust in the application they are using. Research present in the previously quoted paper shows how, despite a great majority of users (81%) recognizes the importance of cybersecurity, less than half of them actually understands what measures to take. While educating users to care enough to make an effort may be useful, preventing some situations altogether is a more effective solution.

Among the most prominent reasons for which security measures are not applied there are time consumption and difficulty of understanding, this clearly shows the need for usable security, namely the need for security functionalities to be actually usable, without impacting regular tasks.

4.1 Overview of UI/UX design concepts for web application security

Without aesthetic, design is either the humdrum repetition of familiar clichés or a wild scramble for novelty. Without aesthetic, the computer is but a mindless speed machine, producing effects without substance, form without relevant content, or content without meaningful form.

- Paul Rand

Every human interaction with a system exists as the intersection of three things: the perception of possibility of the interaction, the reason to perform the interaction and the knowledge on the interaction can be brought to completion. For an interface provided by an application to fulfil its goal those three parts of the interaction should be optimized by the use of three corresponding principles: affordance, authority and familiarity.

4.1.1 Affordance

Don Norman in his "Design of everyday things" writes: "The term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used.[28]"

This design-focused definition is much more technical than the original by Gibson[29], putting the main focus on the specific item to be used. In software UX design, while on one hand all objects are subject to the same constraints (like bidimensionality), on the other hand there is more control on the offered affordances than in a physical object, so properly conceiving how an interface can be interacted with becomes a priority for developers. A digital interface offers more freedom in its creation than a physical object, but has clear limitation in its perceivability, because it hasn't got those "default" properties that physical items have, like weight or texture.

In the specific case of web application security affordance has a double value:

- Preventing a malicious user from doing certain things by "hiding" the opportunity for action.
- Preventing damages to the assets caused by errors of regular users by offering clear affordances for legitimate functionalities.

Both of these objectives are reachable by "punishing" certain uses of the applications, by showing alerts, making buttons turn red or playing some uncomfortable sound, effectively limiting the interpretation one can have of the web application object [30] and by "rewarding" good actions with positive feedback.

If an interactable item does not have a clear affordance even a well-intentioned user might accidentally create a security issue: for example the majority of applications automatically mask passwords or other sensible information while they are being typed, if the visibility switch that is generally offered only shows the information while pressed, but is interpreted as a toggle switch the only negative impact is a bit of frustration for the user, but if the opposite is true the user might forget to hide the data again and could expose it to passersby. Even worse, if not correctly labelled, a switch making an information public could be interpreted as the previous visibility one.

A lot of other examples can be done about buttons used as switches or about the dangers of draggable items (and the identification of the areas in which they can be dragged), but they all bring to the same conclusion: unclear information regarding a function will only cause a misuse of said function.

4.1.2 Authority

"The authority principle refers to a person's tendency to comply with people in positions of authority [...] The authority principle is an example of the human tendency to use judgment heuristics. In this case, the implicit assumption is that those in positions of authority may wield greater wisdom and power, and therefore, complying with them will lead to a favourable result. As humans, we are inclined to make the easier decision rather than the accurate, more effortful one."[31]

For the common user a web application is an authority, even if only in the context of a hypothetical imperative, and in an enterprise setting, where there are consequences for not doing their job, they might feel even more compelled to perform the actions they're told to perform; it's common in phishing attacks that the attacker tries to impersonate some kind of authority. Perceiving a suggestion as an order from an authority can convince people of doing something they don't agree with or know has a negative effect on them or someone else[32]. When creating an interface developers should consider which choices it is suggesting (even implicitly), as influencing decisions can be extremely impactful, especially in enterprise settings, where a wrong security or business decision could be catastrophic. The numeric threshold at which indicators or graphs change colour directly dictates the reaction of users, so it should be defined with a clear knowledge of the situation in mind. Using the authority principle properly means increasing credibility and alleviating the strain of decision-making, so it can be exploited also when there is no malicious intent.

4.1.3 Familiarity

Computer interfaces, in contrast with other "interactables", are expected to be way faster to understand, this depends on a number of factors, like the high number of pages or the fact that users are used to access all of them through the same physical object, this leads to a rising relevance of the concept of familiarity. "A well-practiced task requires fewer cognitive resources to be completed successfully[33]"

In general, performing a well-known task successfully require fewer cognitive resources, because pattern recognition is "cheaper" than understanding a new pattern.

According to Jakob's Law of Web User Experience[34] "users spend most of their time on other websites, not on yours", consequently the patterns they use are likely not the ones specific to a single application, so it's easier to adapt to the current standard than to set a new one. It's important to note that a lot of these patterns did not originate by the interaction with web applications, but exist in human reasoning and might have cultural variations even on the most basic concepts[35].

The mental models generated during this experience completely control normal interactions, to the point where users could ignore entire areas of the screen because they wrongly assume what functionalities those areas provide. "When designs deviate from convention, users may not be able to bridge the Gulf of Execution[28] to see how to approach their tasks in the interface, and thus will make mistakes. Especially during well-practiced processes, they allocate fewer attentional resources and can easily slip and perform the wrong step without realizing it.[33]"

The danger of this mistake is that an error that stems from ignoring an information will not be noticed, thus it can't be corrected; obviously without a correction the error will be repeated, proportionately increasing the damage done. It's the designer's responsibility to prevent the misinterpretation of an application's model, by either being consistent with preexisting mental models or, when necessary, force the learning of new one through unambiguously different execution paths.

This property often overlaps with affordance, especially for those patterns that are learned outside of the digital environment, but concentrates on common practices and application related mental models.

4.2 The impact of UI/UX patterns on web application security

Human behaviour can be influenced through "rewards" and "punishment", even if these are just perceptions deriving from neutral actions, as a consequence an interface response to a user action can heavily impact their next decisions. Designers should keep in mind how they are affecting this behaviour and consider it when they model users and when they collect metrics. Bringing users' attention to one specific item instead of another is enough to make them follow a different flow of execution and learn the corresponding mental model. Interesting research on anthropomorphic perception of interfaces [36] show how even small changes to the textual interaction a machine can have with a user is enough to make it perceivable as more humanlike and consequently improve users' performance and reduce stress; both of these things make users more compliant to security norms and suggested behaviours.

Another thing to consider is the understandability of information: if for some reason, whether it be absence of clarity or limits of the user's perception (due to colour-blind unfriendly colours, items or text too small, etc.), users cannot understand what the application is trying to communicate they will be unable to take the right decision on how to act. Misunderstanding the effect of an action can expose assets belonging to both the users and the company that developed the software even if there is no intention of causing said damage.

In 2020 84 percent of organizations surveyed experienced a security incident caused by a human mistake[37] effectively making the problem not ignorable by anyone. Considering the ever-increasing number of cyber-attacks, it's in companies' best interests to invest in preventing at least the accidental breaches.

Another thing to really consider is that the majority of applications are products meant to be used, but no one would use a product that doesn't seem secure enough in its functionality, so "UI has to give a "sense of security" [38]".

It's important that the users feel secure enough to actually use the application, obviously it would be better if it was a perception instead of a feeling. This can be obtained through the application of simple concepts[38]:

- Consistency: using reliable and consistent UI libraries, that are already tested extensively.
- Product placement and ad transparency: users must be aware of what is an ad and what is not.
- User in control: use of tips, feedbacks for actions and clear settings all help users feel like they have control over the situation, which is needed to feel safe.
- Never ask for more information than needed: GDPR and other regulations adopted should be shown and control on which data are provided must be offered.
- High quality copy: language style must be consistent and match the brand concept.

4.2.1 Design for diversity and accessibility

"Catering for universal design", despite being the second "Golden Rule" of interface design is a vague concept, that needs further explanation in order to be applied. First of all, there are two ways to address the issue: Inclusive Design and Universal Design. Inclusive Design is based on creating specific solutions to satisfy the needs of specific users and then expand those solutions and the benefits they bring to everyone, this can be expensive and really complex to do, even though it has better results, because it requires the involvement of various people representing the diversity for which one is adapting the situation. Universal Design vice versa uses a top-down approach, focusing on just making something that works regardless of possible user impairments, this approach is easier in enterprise settings as most of the time implementing redu ndancy of information through multiple senses is enough to solve the problem. For both approaches following W3C standards[39] is a must.

Providing textual description of images and a tabular visualization for graphs can give visually impaired people the possibility to have a complete perception of the content using screen-reading programs, and so do colour-blind friendly interfaces. Closed captioning any audio content can do the correspondent function for people with hearing limitation. Users with physical disabilities could appreciate an interface navigable using just the keyboard and voice recognition assistive programs generally convert inputs to keyboard commands.

Despite being features inserted to help a small percentage of the user base they can improve everyone's experience, as everyone could temporarily be in a condition of limited perception.

Designing for diversity also includes cultural difference that might impact the perception of information (ordering items from left to right only makes sense if that's the direction in which users are used to scan the page). A paper by A. Alam et al. [40] analyses the ways these differences can affect security, focusing on how the concept of sharing exists in different cultures, which obviously includes sharing of passwords, software or devices. The key finding is that ethnocentrism in design can lead to extreme misuse, so it's important to appropriately study the customs of every user group before deploying a security solution.

4.2.2 Wellbeing at work: contrasting social engineering through satisfaction

A prominent technique in social engineering attacks companies consists in making users feel like what they're doing is not a big deal and that not following the directives (like not opening email attachments) is not going to have negative impact. This approach mainly works because the average worker doesn't really care about the integrity of the system they're working on or at least doesn't care enough to prioritize it in respect to just "doing their job", knowing well that whoever is in charge will blame them if they don't respect deadlines. Creating a better environment for people that use a piece of software is obviously something that's outside of a programmer's responsibility, but this doesn't mean that it cannot be influenced: frustration can create a sort of opposition in users, so it's important to avoid unnecessary complications and possibly "reward" them for a proper use of the application. As expressed in a paper by M.Hertzum [41] if a tool is not immediately usable ("ready to hand") the attention of the user focuses on (understanding) the tool rather than on (completing) the task and "These shifts are associated with frustration and other negative emotions because the breakdown thwarts progress on the task, at least temporarily." The present explanation of the first Kammergard's four perspectives on HCI [42] suggests that interfaces designed for immediate comprehension, that operate as an extension of the user, are the only ones that embody a positive UX; in a context in which people use the same product every day for multiple hours this behaviour is to be expected.

4.2.3 Brand impersonation

In phishing practices it's quite common for attackers to impersonate brands or authorities in order to make victims comply with their orders, so the legitimate owner of the identity. IDS and other similar application have among their functionalities the possibility to send emails with links to internal pages which implies the necessity to secure those emails beyond the protocol level and make them as phishing resistant as possible. Obviously, a collection of third-party tools exists to prevent phishing, but the presence of them on the receiver system is not verifiable nor enforceable and the knowledge needed by users to identify frauds is never guaranteed.

Applications need patterns to avoid phishing through impersonation that do not rely on user's responsibility:

- The mail should be well formed, with high quality copy; malicious emails often present themselves as urgent, so avoiding alarming language is a good practice.
- The mail should contain the official logo of the application to be instantly identifiable.
- All links should be written in their original long form and should be well formed, avoiding special characters and only using natural language words.
- The mail should contain an identifiable contact to the owner of the application (like a support email address) to add an additional element of recognizability and to immediately give the possibility to report incidents.
- Using information not known to external parties can decrease the replicability of a mail, a contract id is perfect for this function.

The application designers cannot educate end users, but introducing a small warning asking them to check the legitimacy of the message through a few simple steps could make them notice manipulated emails. Considering that in the past a relevant number of malwares has been spread using malevolent emails at least limiting this intrusion pathway would reduce the pressure on technical security systems.

4.2.4 Passwords

In order to prevent security issues, one must consider the impact of the human factor and that no security functionality is useful if it's used improperly. A NNGroup's article that has already been quoted before[25] explains some cases in which this happens: "If you require an email address as the user id. In many cases, users assume you are asking for their email address and their AOL password."

First thing to consider is the need to clarify which credentials are being asked, now, more than when the article was written, users own a lot of accounts for a lot of different services and these services are often interconnected or overlapping. "You should always recommend that they choose a new and different password"

However, password reuse is still a big problem, so the suggestion to choose a unique one keeps being valid. “If your rules are too strict, many users will not be able to use names and passwords that make sense to them. This increases the likelihood of users forgetting their login information the next time they visit. Forgotten passwords are the cause of countless repeated registrations across the Web.”

An excessively complicated password means users will either write it down (incredibly dangerous in an office) or just forget it, with all the problems related to password recovery and re-registrations. “You should place instructions for user ids and passwords immediately next to the field label.” Also, any instruction related to how credentials must be formulated has to be written close to the input fields or it will probably just be ignored.

The password problem is as old as internet and yet to be solved, in spite of all the partial solutions that were found. On smartphones the solution was found in graphic sequences, easy to remember but random enough, anyway this is a type of input that is just not feasible for PCs. Nowadays they rely on fingerprints and face recognition, but while it makes sense to give those data to a device you own, it definitely feels too much to give this kind of information to a random site on the internet and in the eventuality of compromised credentials the recovery it’s really complicated. The privacy problem might be solved if the personal information (face/fingerprint) never leave the device, but proving it to a regular user might be a challenge. Things like third party login only make sense in an enterprise setting, unless the third party is the organization itself, offering logins for other services, this is doable only for some organizations and only for some services. Mobile logins like QR scanning and OTPs can partially fill the gap, but have a higher interaction cost, also, they have the same problem of third-party login.

“For sensitive systems, many users feel more comfortable when they see an explicit logout button. For most systems, however, you can assume that users will not log out and instead will simply leave”

A decent UI should assume that users can behave very differently and act to guarantee security accordingly, some users may feel the need to manually logout every time, but others will simply close the page, creating the need for a timeout system.

Chapter 5: Developed guidelines

A literature survey by D. Jacobs and T. McDaniel[43] shows that misconceptions are the most analysed theme in regard of user-centred security, yet there is a minimal number of guidelines on how to avoid them. The previously quoted research presents how "the motivation for risky behaviours stemmed from misconceptions, lack of knowledge, or design flaws. Users engage in risky behaviours when the security and privacy solutions are not friendly". The main way UI/UX can prevent security problems is by preventing user errors, exposing clear affordances and reducing frustration, this can be done at different levels and in different ways.

5.1 How to approach the following guidelines

The guidelines present in this chapter have been developed in the context of the user interface of aizoOn's IDS, have been tested on that application and are formulated for that specific situation, regardless, they are still appropriate for any safety critical enterprise application and can easily be adapted to any user interface. IBM's Carbon Design System Guidelines were taken into considerations as IDS mainly uses those versions of React components, with some minor differences.

The guidelines are meant to be read by both designers and front-end developers; a choice taken because of how frequently the decision making of the two overlaps in most development studios. Each section is divided in "Theory" and "Practice", respectively describing the design principles to be applied in the relative situation and the practical suggestion for the developers. The "Theory" sub-sections are mostly oriented toward designers and try to give a general idea of how usability problems could impact an application security, the "Practice" sub-sections on the contrary describe in detail how to avoid the aforementioned usability issues.

When applying the guidelines to an existing interface the severity of each violation should be decided by the evaluator based on the context in which is found. This decision has been taken because the severity of security dangers of an interface problem does not depend on the nature of the problem, but on the effect of that specific interaction, making it impossible to define a priori.

5.2 Enumeration of guidelines

5.2.1 Colours

Theory

Conventions

Choice of colours can greatly influence the experience provided by an application; specific colours have implicit meanings users are used to. Colours that heavily depend on conventions to be interpreted, like primary green for confirmation, primary yellow for warnings and primary red for errors or deletion, should be only used for those functions and “neutral” colours should not be used for interactables. Bright colours that stand out attract attention, so they should be limited to calls to action, otherwise users’ page scanning flow will be hindered by unnecessary alerting items that could have them ignore actual points of interest.

Interpretation of colours obviously depend on cultural factors, but when they are meant to represent more complex ideas the meaning could be completely different, so designers should always consider the cultural subtext of the target users.

Designers should try to keep the number of colours used as low as possible, otherwise the interface will feel distracting and overwhelming, with a higher number of colours it becomes progressively harder to form a satisfying palette.

Identity

Brand/product identity is really important in recognizing an interface and its affordance, so colours chosen should properly reflect it, without conflicting with common conventions. Choosing an aesthetically pleasing and meaningful palette could collide with what is dictated by conventions, in those cases it’s important to remember that no amount of branding by one company is able to change preconceptions built over years by every other existing product, so the only viable solution is to find a workaround, either by not using the problematic colours for misunderstandable functions or by changing them: a change in tone or brightness in order to obtain a different shade of the chosen colour can often solve the issue.

Colours that cause the aforementioned misunderstandings are the one strictly linked to specific concepts, like the classic semaphore of green for safety/correctness, yellow for low danger/warnings and red for high danger/errors.

Accessibility and colour relationships

Something that directly impacts the readability of a page is the relationship of adjacent colours: if the contrast between text (or any generic item) and its background is not enough it will just not be perceivable by users. On the other hand, high contrast is what makes a particular colour stand out, so despite not being a problem to legibility it might attract attention to the wrong part of the interface.

Colour-blind people have an additional problem in differentiating some colours: the whole application and especially charts a colour-blind option should be offered when the chosen palette is not already colour blind friendly.

Depending on the type of colour blindness designers should conceive a proper translation of the semaphore, recognizing which option could keep the same meanings, in general, warmer colours are associated with dangers and colder colours with safety. Sometimes the middle option is problematic because of the lack of a third perceivable colour, using a lighter version of one of the other two is often a better option than choosing a blend of them, as it could just appear grey, losing any subtext it could have. Considering the existence of total colour blindness also guaranteeing a different brightness for each colour is needed and using mono-coloured scales is a viable possibility.

Implementing automated accessibility testing in the development process is strongly suggested.

These guidelines refer to the most recent Delta E standard (dE00)[44] by International Commission on Illumination[45] to measure colour difference, which, despite having discontinuities, is the most appropriate scale usable.

Practice

1. The application's interface should use the smallest possible number of colours.
2. No information should be conveyed exclusively with colour.
3. #00FF00 green, #FFFF00 yellow and #FF0000 red and colours with a Delta E difference smaller than 30 points from them should only be used for success, warnings and dangers respectively.
4. Non-text items should have a contrast ratio of at least 3:1 against adjacent colours, as per WCAG 2.1 [46].
5. Text should have a contrast ratio of at least 4.5:1, reduceable to 3:1 for text with at least 18 points in size or 14 points in size if bold.
6. The interface should be colour-blind friendly or offer an option for colour-blindness support.
7. Primary interaction colour should be a high contrast bright colour.
8. The correct visualization of colours should always be verified via proper testing.

5.2.2 Structure

Theory

Item position and size

Positioning of information can have a great impact on how the information is perceived. Users tend to scan the page from left to right (at least in the western world) and from top to bottom, so contents of the page have to respect that order to correctly represent priority. Users usually scan web pages in a "F" shaped pattern[47] or focusing on headers and sub-headers if the page layout is efficient and the titles explicative (this is called "layer-cake" pattern[48]). In general, the second pattern is more effective for the users, so designers should strive to make it the first option, this can be done by making the sub-headings stand out and by positioning the relative text bodies in a way that indicates to which heading they refer to, following the Gestalt's principle of proximity. Dividing the page in chunks is a good idea even when it's not a text page.

Furthermore, even if an enterprise application has no advertisement banners, users still experience banner blindness, because of the patterns they learned in everyday use. Designers should keep the main content in the centre of the page and avoid putting relevant content on the sides or in the very bottom. Navigation is usually expected to be found at the start of the page, which means either at the top or on the left side. Also, keeping items with similar functionalities in the same position for different pages can reduce the cognitive effort needed to find the points of interest in the page.

As position does the size of an item also impacts its perceivability, items too sm all will just not be seen by users, while items too big will feel overwhelming and could overflow the window size.

Consistency, inside the application and with common practices, is a principle that should also be followed when positioning items.

Users might use application through different screen sizes (or just different window sizes) so it's important for the interface not to lose its structure and proportions when these sizes change; every item in the page should be adaptive. On the other hand, item moving or changing in size independently from user input will remove the feeling of control needed to properly operate an application and might lead to clicking errors, with consequent danger for the system.

Item Identification

The position of items can convey meaning about that item's role, but in complex pages it might not be enough: every non-textual item should have some unique and unambiguous textual identification. Headers and sub-headers should be close enough to their relative content to not create any misunderstandings, to better clarify grouping of items using container cards is suggested and confirmation and cancellation buttons should always be positioned after all the content they refer to.

These measures serve to make users' understanding of a page's functionality more immediate and consistent with their mental models, failing to satisfy these conditions means creating a confusing and misunderstandable interface.

Item Relationships

When dividing a page in chunks any difference in size should represent a big difference in importance and this is also true for simple items. Relative size of items and their containers should reflect their importance on the page. Designers should try to divide the interface in equally sized sections for similar content, at least in the direction perpendicular to the one of scrolling.

Items should have enough distance between them to be easily distinguished and avoid overlapping in the untouched state of the page. Two or more items on the page can overlap exclusively when the one upfront is the only relevant one and the others do not bear any useful information or interaction, this should be accompanied by changing the focus to the relevant item, blurring or disabling the others. When making a decision all the options have to be taken into account, so multiple alternatives, even when mutually exclusive should not overlap or hide each other under any condition.

Practice

1. Main content should be shown in the centre of the page.
2. Navigation options should be shown on top of left side of the page.
3. Each page containing content relative to different tasks should be divided in one block for each task.
4. Every block should have a corresponding header.
5. All non-textual items should have simple unambiguous textual description of their function, either inside or in their proximity. Additional instruction can be placed either immediately under the item or shown when the item is selected or being hovered on, but before it's clicked.
6. More relevant items should be shown first going from top to bottom and from left to right (for users that read in that direction).
7. If there is no relevance difference between two items they should be ordered alphabetically by name of descriptor.
8. More relevant items should be bigger in size.
9. Items of the same importance should be of the same size, especially if they represent multiple choices.

10. Items in scrollable parts of the page should have the same size in the direction perpendicular to scrolling.
11. Every item should be adaptive and its visualization on different window sizes should be verified.
12. Items should not change size or position without user input.
13. Items with equivalent function in different page should have the same position if possible.
14. Items should have enough distance between them to be easily distinguished.
15. Items should avoid overlapping in the untouched state of the page.
16. Multiple alternatives should not overlap or hide each other under any condition.
17. Two or more items on the page can overlap exclusively when the one upfront is the only relevant one and the others do not bear any useful information or interaction.
18. Items should not overflow their container.

5.2.3 Content

Theory

Textual Content

Textual content is the main vehicle of information in application, meaning that investing attention in making it as clear and comprehensible as possible can prevent users from taking wrong action based on the information they understand.

Firstly, text must be physically readable: all textual content should comply with WCAG, both for size and contrast, as previously stated and the chosen font should be simple and neutral, avoiding elaborated fonts. The font used should be unified for all equivalent content and belonging to the same family across all the application, to create consistency. It's appropriate to use different fonts of the same family for specific types of text, like pieces of code or tooltips. The chosen font will influence the perceived tone of the text, so for "colder" enterprise application it's always better to use more formal fonts, opposed to the "decorated" fonts the flexibility of a more conversational everyday use application may allow. In the case of IDS "IBM Plex" font family is used.

The second point to take into account is the language used, that should be clear and unambiguous and the tone should be coherent with the context in which is used. The language should be user-centric focusing on the effect of an event or action on the user, instead of on details them[49], this is especially true for instructions. If the web application is showing technical information, shortening titles as much as possible, without oversimplifying content, and providing summaries (or some summarizing data) can improve the readability. Text should include a clear definition for each non-statistical information given.

Headers, sub-headers, tooltips and other descriptive textual content should be visualized as close as possible to what they are describing and use the smallest number of words needed to express their meaning unambiguously, to achieve maximum effectiveness. As previously stated, header's size should reflect hierarchy.

For obvious reasons text need to be grammatically correct and follow the rules of the natural language in which is written.

Order of information should follow its priority.

Icons

For the sake of consistency designers should use the same icon pack for all the application. IDS uses IBM's Carbon icons so "16px and 20px icons are optimized to feel balanced when paired with 14pt and 16pt IBM Plex", but for different packs and fonts this pairing may differ. In general icons slightly bigger than the chosen font contribute to a satisfying feeling.

Icons can be useful for immediately recognizing an item's function, but must not be overused, if an icon has a clear meaning it can provide value, otherwise it will just create confusion and should be removed. Recognition is better than recall.

Graphs and tables

Graphs ca be useful to show numeric data in a simple and concise way, but hey can present a challenge for people with limited perception: all graphs should be colour blind friendly, have a clear header describing what they represent and a caption with details needed to interpret it. Data contained in graphs should also be presented in a tabular form, either as an addition or as an alternative to the graphical representation.

Tables, despite being compact, always risk being overwhelming at first sight, to diminish this effect they should have the smallest number of columns needed to deliver all the information, condensing mutually exclusive options when possible. Column titles should be short and clear.

Practice

1. All text in the interface should belong to the same font family.
2. Apart from specific bit of text like pieces of code all text should be in the same font.

3. All text should be big enough to be easily readable.
4. All text should be grammatically correct.
5. All text should be clear and unambiguous.
6. Header size should reflect hierarchy.
7. Descriptive text should be precise and use the smallest number of words needed to express the concept.
8. Descriptive text should be visualized as close as possible to what it is describing.
9. Instructions for the user should focus on the effect of their actions.
10. Order of information should reflect its priority.
11. Text should include a clear definition for each non-statistical information given.
12. All icons used should belong to the same icon pack and style.
13. All icons should have a clear meaning.
14. An icon should keep the same meaning in all the application.
15. If multiple items belong to the same group either all of them should have an icon or none of them should.
16. Icons should be accompanied by descriptive text whenever possible (even just as a pop-up tooltip) and can be used alone only when the icon has a universal meaning.
17. Icon size should be proportionate to the relative text.
18. All graphs should have a header describing what they represent.
19. All graphs should have a caption with details needed to interpret it.
20. All graphs should be colour-blind friendly.
21. All graphs should have a tabular representation available.
22. Tables should not have redundant columns.
23. Table columns representing mutually exclusive information should be condensed into one.
24. Table columns’ titles should be descriptive and compact.

5.2.4 Interactables

One of the main concerns for web application security is constituted by user's textual inputs, in fact injections are possible if those inputs are not sanitized. However, interactions by users can happen in many ways and each of them has to be handled properly to avoid security problems. Human computer interaction literature usually defines 5 interaction styles[50], choosing which one to use for a particular interaction can be really impactful from a usability point of view, but regardless of that choice some measures must be applied to prevent misuse:

1. Direct manipulation: users need to be able to understand where they are dragging items in real time to correctly position them, so real time feedback on what is being changed in the system state is essential. If this continuous representation cannot be provided it's better to resort to other methods. Also, correctly signalling what can be manipulated can prevent frustration or wrong interactions with other items. Technically even using a mouse is a form of direct manipulation.

2. Menu selection: provides structured decision-making, this means that designers are responsible of how this structure is composed. Users have a "first-fit" approach, so the order in which the options are shown has direct effect on user's actions. If some options are not available or would cause errors in that moment's state they should be disabled (or removed completely), this is particularly important for options that depend on privilege.

3. Form fill-in: the use of classic textual input obviously implies the need for validation and sanification, but, beyond that, an incorrect interpretation of what the user is being asked can make them invertedly give wrongful or sensible information. Every field should have an appropriate label and constraints to what can be inserted should be visible immediately next to the field. If a part of the form needs to be compiled with information correlated to what is asked in another part putting the two parts on different pages forces user to go back or memorize information to not commit errors, to avoid this unnecessary effort the whole form should be kept in the same page whenever possible or at least grouped by independent sections of itself.

4. Command language: it's slowly falling out of use and it's nearly absent from web applications, it requires recall instead of recognition so it's unusable for untrained users. Seen its complexity one should be really careful of not allowing dangerous actions with wrong commands or avoid it if possible.

5. Natural language: with the increase in the diffusion of artificial intelligences and personal assistants, natural language inputs are now used in a lot of applications, aside from the input sanification needed for all language inputs, this method requires additional controls due to its inherent unpredictability and risk of misunderstanding, operations

executed as a response to natural language inputs without proper confirmation might cause unwanted behaviour.

Developers should expose the right affordances to let users know what kind of interactable they are dealing with and always give visibility of the current system status. Every interaction should have appropriate feedback, with explicit alerts if needed. Warnings and error messages should be in close proximity of the relative interactable and suggest what to correct if possible.

To prevent accidental clicks or unresponsiveness clickable area should correspond exactly with the interactable item. In order to prevent annoyance, interacting with an item should never change the current page's layout, if different items are needed after the change of a state a new page is needed.

Keeping the number of clicks needed for an interaction as low as possible can also greatly reduce the possibility of errors.

Ambiguity in the type of interaction requested will also cause misuse with possibly catastrophic consequence, making it essential to use each interactable item for the function it is supposed to carry out.

Practice

1. All user interactions should have feedback.
2. All potentially dangerous or irreversible actions should ask for confirmation.
3. All potentially dangerous or irreversible actions should have appropriate warnings beforehand.
4. Textboxes should have a title and a placeholder.
5. Fields with mandatory input should have a clear tell, like an asterisk referring to a note somewhere of the page.
6. Mandatory textboxes should show an error if committed while empty.
7. Invalid input should be checked in the front-end and notified in the interface.
8. If some textbox has a specific requirement, it should be specified beforehand with a warning close to the textbox.
9. If a textbox only accepts a limited number of options, it should be replaced with a dropdown menu.
10. Buttons should change to a darker colour when the mouse hovers over them and they should give immediate feedback when pressed, accompanied with colour inversion.

11. Buttons for confirmations or active actions should be of a bright, noticeable colour.
12. Buttons to cancel actions or delete information should either be black or red.
13. Buttons should never be used as switches.
14. Inactive buttons should be of a lighter, less vibrant colour, and have no reaction when hovered or clicked.
15. Inactive interactables should have in their proximity the information of why they are inactive.
16. Dropdown menus should have a placeholder as the preselected option if all options are equivalently important or the safest selectable option if not.
17. Dropdown menus should not cover any information relevant for selecting an option.
18. Switches should have a clear description of what is their purpose and to which field they are related if any.
19. The state of a switch should be clear at any time.
20. Switches related to other fields should be close enough to avoid any confusion.
21. Forms should be entirely on one page, if not possible they must be divided in a way no information is needed to be recalled from one page to another.

5.2.5 Navigation and task execution

Theory

Every application's functionality should be accessible by navigating to the proper page at any given moment, quickly and by everyone. If some specific tasks should be performed in order to access a functionality, then those tasks must be accessible from the same page of the aforementioned functionality.

Every similar function should be executable through the same kind of interaction. Users consciously interpret differences in aesthetic as differences in function. Ideally if two similar operations require multiple passages the two sequences of actions should be the same. Learning one procedure is faster than learning multiple and a small difference in similar procedures can lead to distraction errors, especially if it's something performed repetitively.

Everything in an application should conform to established conventions of web application in order to avoid errors. Users are used to operate on other applications, so they learn a series of mental models they expect to find again, failing to meet this expectation means creating an error prone interface.

Practice

1. Navigation menus should present options in an order that follows priority.
2. If an option is not available to a specific user the option should not be shown to that user.
3. If an option is not available due to a series of conditions not being satisfied the option should be disabled and the user should be warned of which conditions are not being satisfied.
4. All main pages should be accessible with one click.
5. Similar tasks should have similar execution structure.
6. Tasks should require a thought process consistent with users' mental models.

5.3 Old IDS and its compliance with guidelines

IDS is an application meant to analyse the network on which is applied, recognize potentially dangerous behaviours using artificial intelligence and provide the relative information to security experts, that will then take action as a response using network configuration functionalities of the application itself. For these reasons the way the data is presented and the interface is interacted with could potentially cause security issues to the user's system. In this section IDS's interface is confronted with what is suggested by the previously stipulated guidelines to verify the frequency and severity of non-compliant behaviours: a sample of pages and components will be shown.

5.3.1 Dashboard heatmap

The home page of the application presents a dashboard with general information about the system and the machines in it. One of the main components of this dashboard is a heatmap showing the risk levels with the passing of time, to correctly convey the meaning this heatmap must accessible and understandable to anyone.

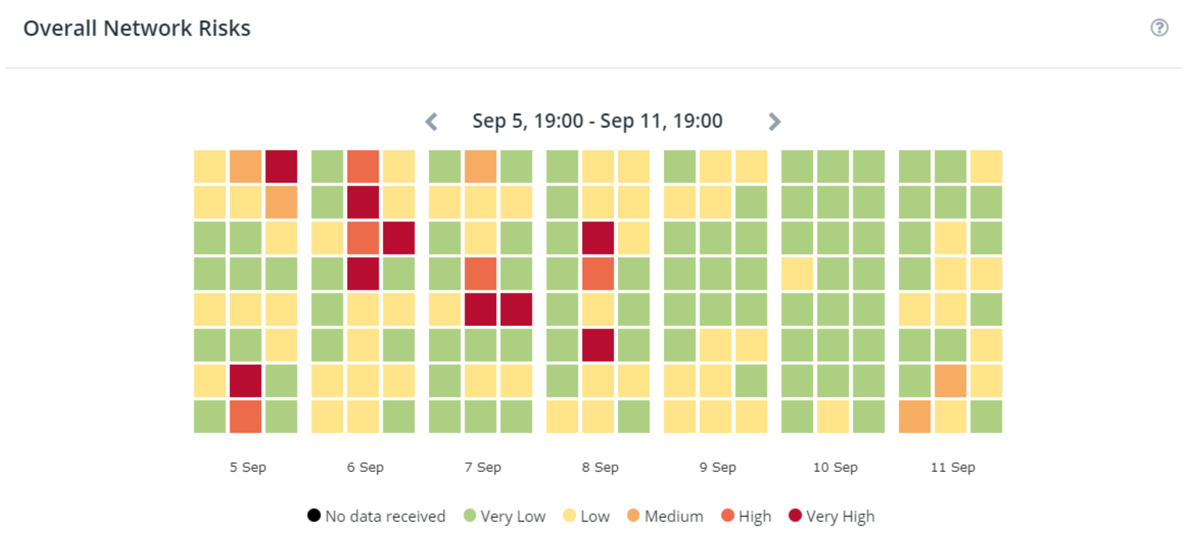


Figure 1: *Network Risk Heatmap, normal vision*

When perceived by a person without any vision deficiency the heatmap presents no problems, the interpretation of colours is consistent with what is explained in the caption and the security semaphore meaning is immediate even without it.

For users with vision deficiency though some problems start to arise: first of all, guideline "*Content 21: All graphs should have a tabular representation available*" is not satisfied as that representation is not present, second there are a series of issues for colour-blind people.

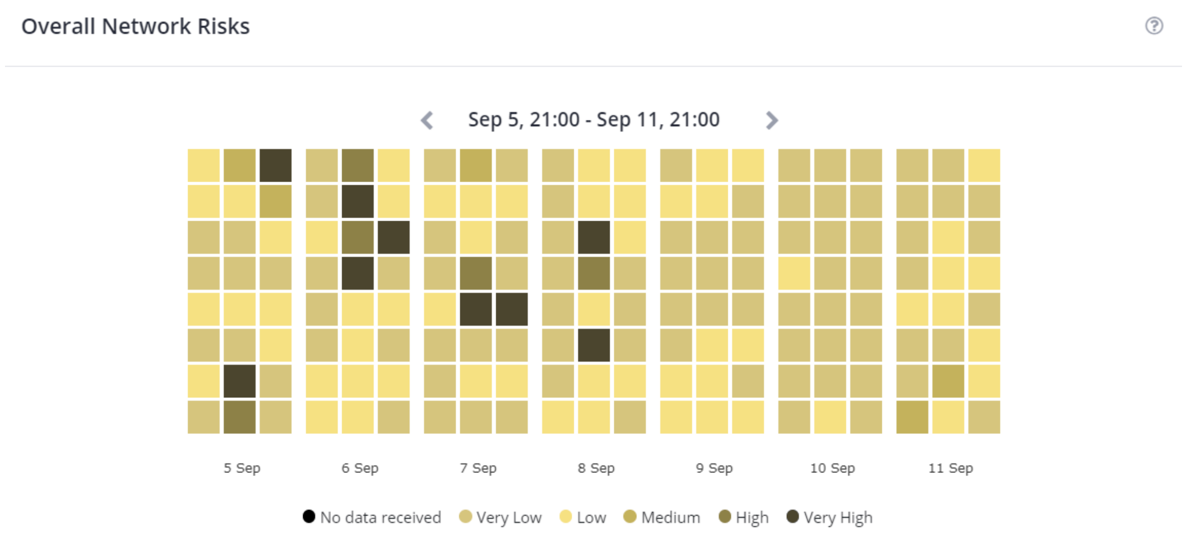


Figure 2: *Network Risk Heatmap, colour-blind vision (protanopia)*

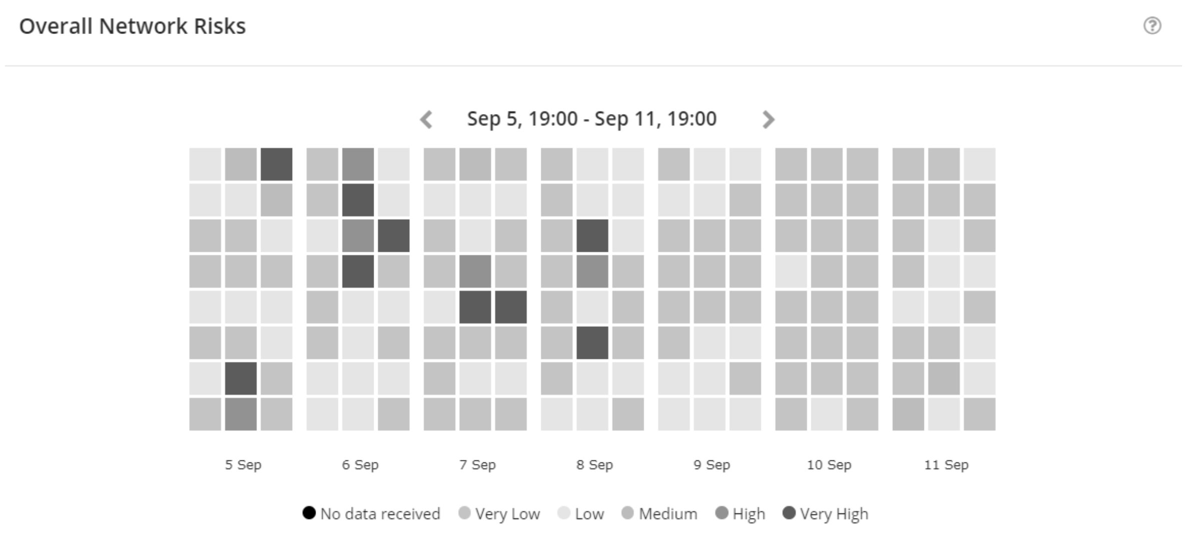


Figure 3: *Network Risk Heatmap, colour-blind vision (achromatopsia)*

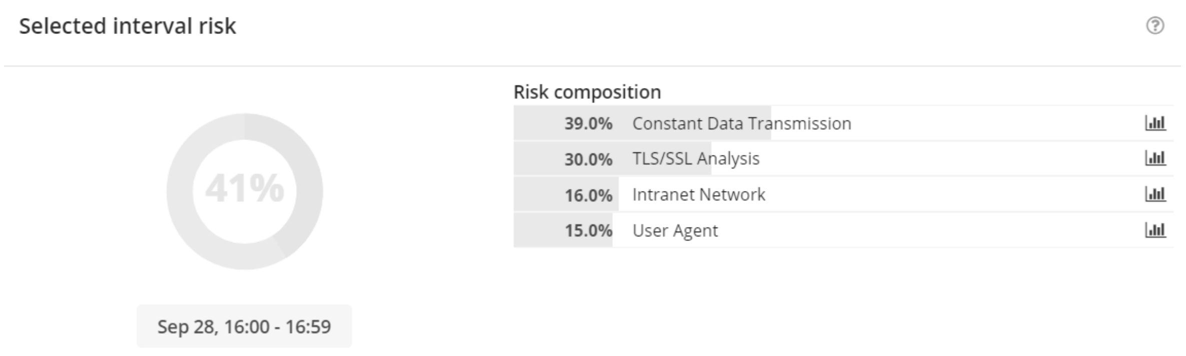
In the case of protanopia (absence of red) and some other types of colour-blindness the guideline "*Content 20: All graphs should be colour-blind friendly*" is not completely respected: as can be seen in *Figure 2* the security semaphore immediately feels inconsistent because "very low" (RGB: 214,197,125) and "medium" (RGB: 196,178,92) risk are both darker than "low" risk, furthermore the two colours are hard to distinguish when not close together, having a Delta E of just 5.824 points, which, despite being considered "perceivable at a glance" is not enough for small and not easily confrontable items like the squares used in the heatmap. For deuteranopia (absence of green) the difference is slightly better, but the semaphore problem still exists, while for tritanopia (absence of blue) there aren't any specific accessibility issues.

For total colour-blindness, as can be seen from *Figure 3* "very low" (RGB: 196,196,196) and "medium" (RGB: 188,188,188) risk only have a Delta E of 2.0642, barely perceptible by human eye and definitely not perceptible for small items like in this case. Additionally, the contrast between "low" risk and the background is of just 1.25:1, too low for guideline *"Colours 4: Non-text items should have a contrast ratio of at least 3:1 against adjacent colours"*.

This absence of support could affect the decision making of a user with vision deficiencies, who could wrongly overlook medium risk situations because of the colour similarity. Despite being a minor problem, as by clicking on a specific square in the heatmap other components on the page show the numeric value of the risk percentage, this issue could still have a serious impact on security.

5.3.2 Dashboard selected risk

As previously mentioned, when selecting a specific interval of time or a specific IP address on the network a series of details about it are shown in other components, this partially compensates the accessibility limitation of the heatmap, but is not devoid of its own problems.


Selected interval risk

41%

Sep 28, 16:00 - 16:59

Risk composition

39.0%	Constant Data Transmission
30.0%	TLS/SSL Analysis
16.0%	Intranet Network
15.0%	User Agent

Figure 4: *Selected interval risk, colour-blind vision (achromatopsia)*

Starting from the “selected interval risk” infographic in *Figure 4* the persistence of the colour accessibility problem can immediately be seen. The active part of the circular chart, in this case representing low risk, is barely distinguishable from the base colour: their Delta E is of only 1.0818 points, barely over the limit of human perception, and their contrast is of 1.04:1. Furthermore the numeric value that could make the information accessible only has a contrast of 1.2:1, not compliant with the guideline “Colour 5: Text should have a contrast ratio of at least 4.5:1, reduceable to 3:1 for text with at least 18 point or 14 point bold size”, and thus is not correctly perceivable.

In this case, even if only for a small number of users (achromatopsia affects 1 in 30,000 people[51]), the non-compliant behaviour could completely prevent the perception of an information, making it an high severity source of errors.

Apart from colour-related issues this section of the page also presents some adaptiveness problems that, while not too severe, involve all users.

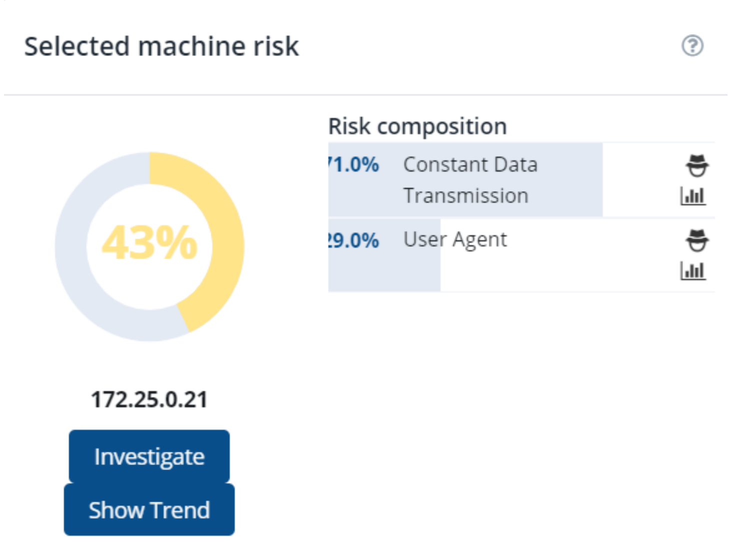


Figure 5: *Selected machine risk, smaller window*

In *Figure 5* it can be seen how, for smaller window sizes, part of the data in the "risk composition" section gets lost due to an overflow of the textbox from its container and how the "investigate" and "show trend" buttons partially overlap. Obviously, both of these issues can be resolved by just enlarging the window used, but for a variety of reasons users might need to have some other software on the screen at the same time of IDS. Guidelines "Structure 14: Items should have enough distance between them to be easily distinguished" and "Structure 18: Items should not overflow their container" not being followed causes a partial loss of information, in the first case, and a risk of wrong input, in the second. The loss of information is reduced by the graphical support (colour-blind accessible, as per Figure 4) and quickly noticeable, making it a low impact dysfunctionality. Similarly, the effect of a click on the wrong button due to the overlapping is limited, as they are just for navigation, still depriving users of the feeling of control or causing a loss of time while they perform a cognitively consuming task could induce frustration and distraction, two conditions that easily lead to errors.

5.3.3 Cognitive charts

IDS presents a series of cognitive charts to help users understand the details of what is happening on the network, these charts can be of great use in reducing the cognitive load user might have to withstand when operating on a large amount of data and in making the execution of task on the application much more intuitive, but for this reason they lose their value if non properly presented.

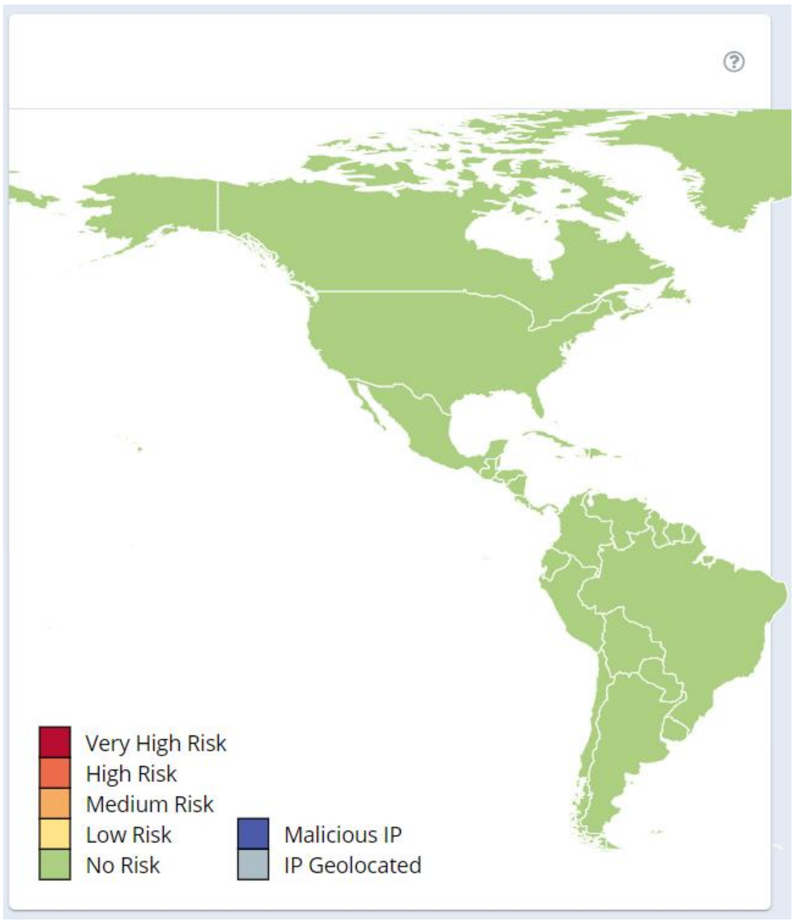


Figure 6: *Cognitive charts, IP geolocation*

Being able to immediately see the geolocation of different IP addresses that interact with the local network can help identifying potential threats, unfortunately for smaller screen sizes the map used does not adapt, overflowing its container and making it impossible to access a part of it. As shown in Figure 6 this violation of "Structure 11: Every item should be adaptive and its visualization on different window sizes should be verified" and the consequent "Structure 18: Items should not overflow their container" is moderately severe, because users can notice and fix it without much effort, but the fact that the extent of the information hidden is not known a priori make it more dangerous than the previous case.

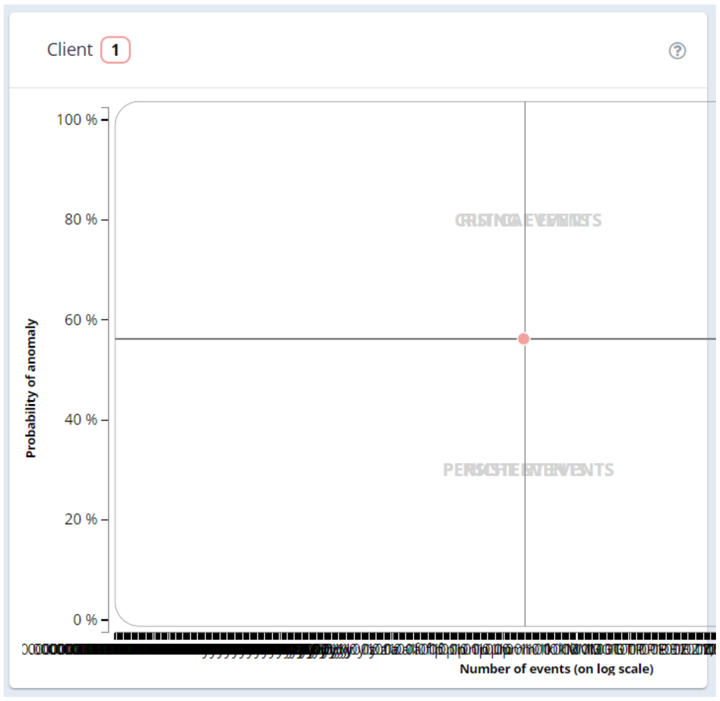
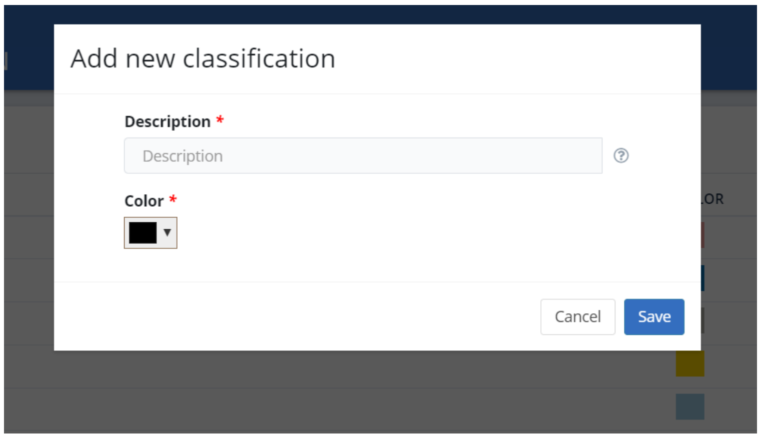


Figure 7: *Cognitive charts, scheduled operations*

The chart in *Figure 7* shows the probability of anomalies in respect to the number of events, when the field of view is increased, though, the indexes on the x axis and the information contained end up collapsing on each other, making the chart unreadable. While the need of this extreme change of dimension is unlikely in everyday use of the application, this violation of "*Structure 17: Two or more items on the page can overlap exclusively when the one upfront is the only relevant one and the others do not bear any useful information or interaction*" completely prevents one specific visualization. The nearly absent probability of this situation is enough to make it a low severity violation.

5.3.4 Network configuration


Add new classification

Description *

Description

Color *

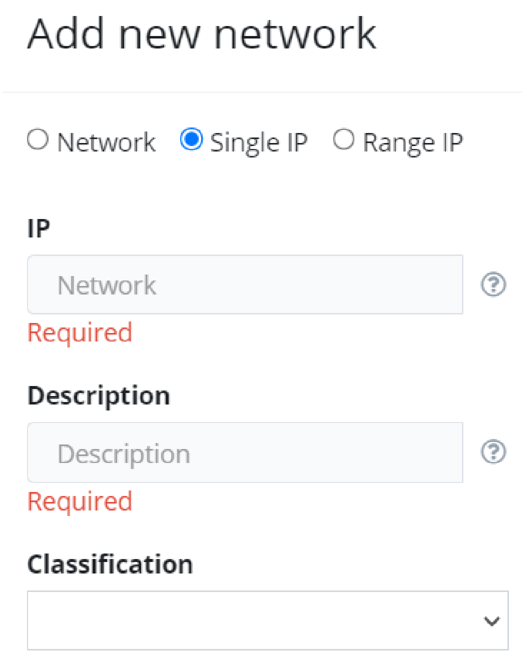
Cancel
Save

Figure 8: *Network configuration, new classification dialog*

Network configuration pages give access to the most impactful functionalities of the application, giving users the possibility to directly operate on the network; a wrong interpretation of an interactable in this section of IDS can directly damage the security of the system.

In *Figure 8* the dialog used to add a new classification to the network contains two mandatory fields and this is conveyed to users with a red asterisk on top of each field, while this is a common practice it's not the only possible meaning it could have. The guideline "*Interactables 5: Fields with mandatory input should have a clear tell*" exist to prevent this ambiguity. After committing an empty field, the compulsoriness of the fields is manifested with a proper error message, but forcing user to commit an error provides a negative UX.

This issue is present in multiple textboxes across the application.


Add new network

Network [✓ Single IP] Range IP

IP

Network

Required

Description

Description

Required

Classification

Figure 9: *Network configuration, new network dialog*

When adding a new network users face a similar problem, the dialog shown in *Figure 9* present some inconsistencies: first of all, in this case mandatory fields are identified by a warning under themselves, unlike in the other dialogs of the application, then the classification dropdown has no "required" warning, despite being compulsory, and no placeholder, reinforcing the idea of it not being a required field. "*Interactables 16: Dropdown menus should have a placeholder as the preselected option if all options are equivalently important or the safest selectable option if not*" and the previously mentioned "*Interactables 5*" not being respected, like for the "new classification" dialog, have no direct impact on security, because the input is prevented if not appropriate, but can create needless frustration in users.

5.3.5 Server configuration

Server configuration pages, like the aforementioned Network configuration ones, can be sensible to user's errors. The form to add a new certificate for example present an inconsistency like its network equivalent.

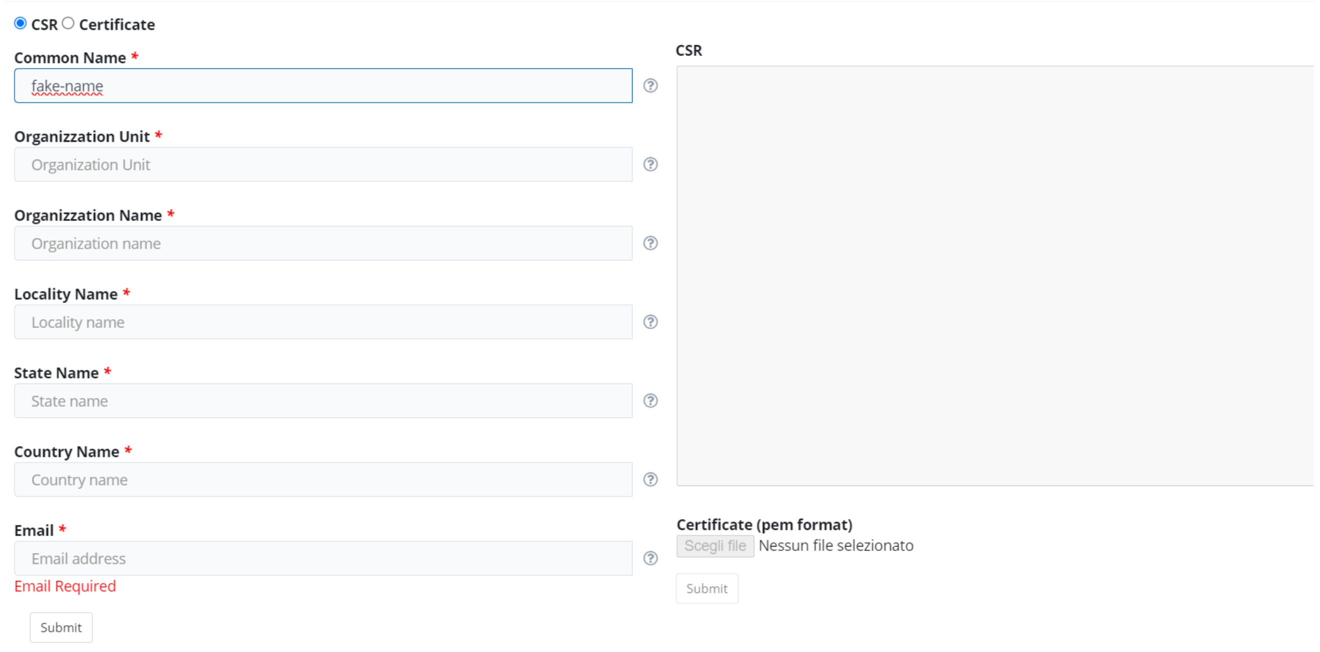


Figure 10: *Server configuration, new certificate form*

As shown in *Figure 10* this form also presents a second part (on the right side) that is disabled until the first part is completed, however this requirement is not specified anywhere on the page and the "CSR" field is not even clearly disable. "*Interactables 15: Inactive interactables should have in their proximity the information of why they are inactive*" not being followed is a moderate violation that can cause confusion in users and have them assume something is not working; whatever solutions user might come up with when trying to solve the problem could damage the system with unnecessary changes to configurations.

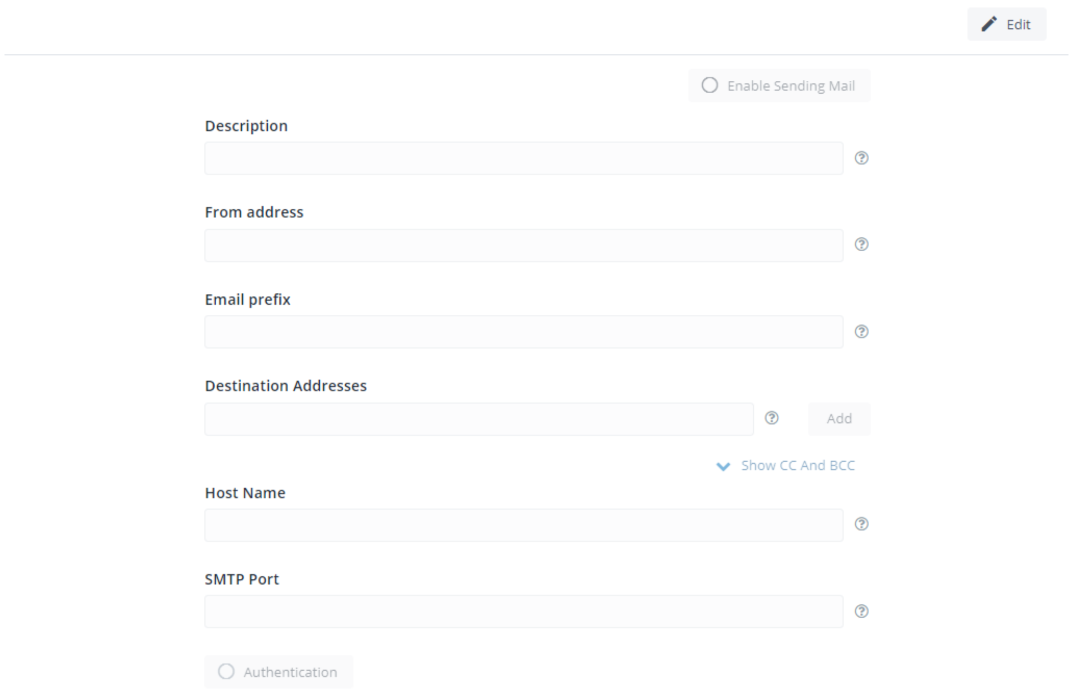


Figure 11: *Server configuration, network alerts (disabled editing)*

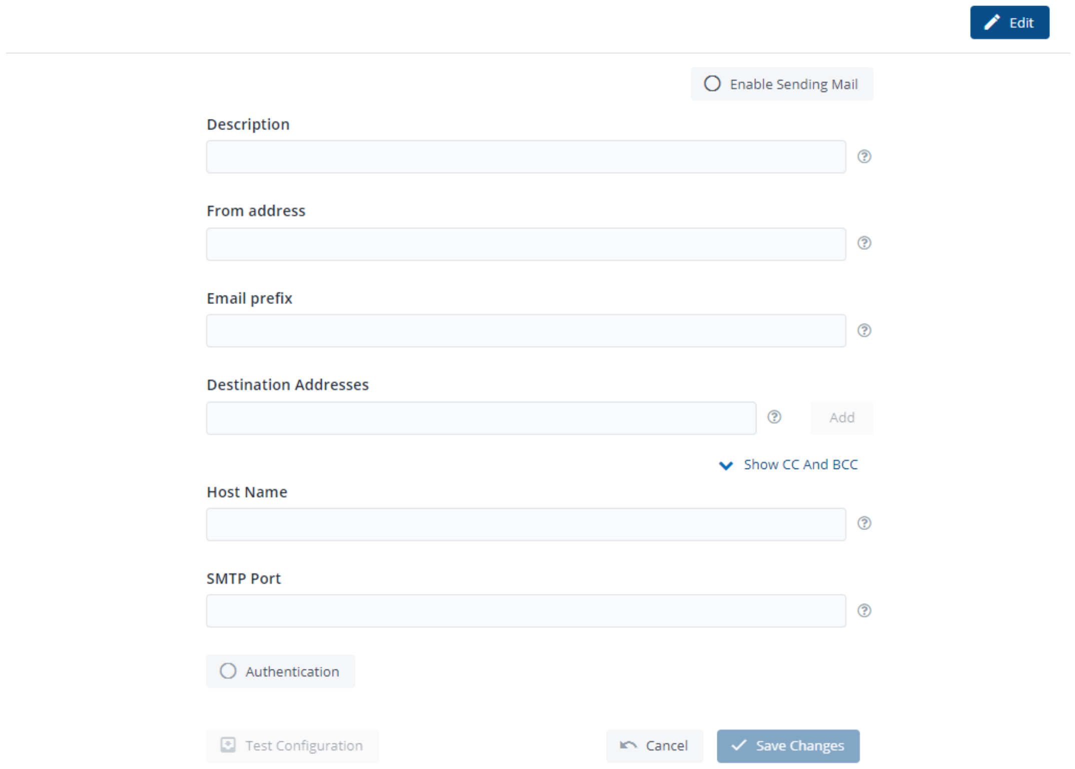


Figure 12: *Server configuration, network alerts (enabled editing)*

Figure 11 and *Figure 12* contain the two states of the “network alerts” form and also the most dangerous violations of the guidelines found in the application:

- The “edit” button acts like a switch, enabling modification of the underlying fields, but looks disabled because of its colour. This is counterintuitive and goes against multiple guidelines like “*Interactables 11: Buttons for confirmations or active actions should be of a bright, noticeable colour*”, “*Interactable 13: Buttons should never be used as switches*” and “*Interactables 19: The state of a switch should be clear at any time*”.
- The “edit” button is in an “active” state while the editing is enabled, yet when pressed it just reverts the fields back to their non interactive state, keeping the information inserted, but not actually committing it. The guideline “*Structure 5: All non-textual items should have simple unambiguous textual description of their function, either inside or in their proximity*” is not followed.
- The disappearance of the “save changes” button prevents user from confirming their input without going back to the editable state, but that button stays disabled without specifying the reason until the whole form is filled with appropriate inputs. Guidelines “*Interactables 15: Inactive interactables should have in their proximity the information of why they are inactive*” and “*Navigation and task execution 6: Tasks should require a thought process consistent with users’ mental models*” are not followed.

The number of issues, the fact that the non-compliant behaviours cannot be separated and the sensibility of what is being operated on make this situation a high severity violation, with a series of possible negative consequences for the system’s security.

5.3.6 Navigation

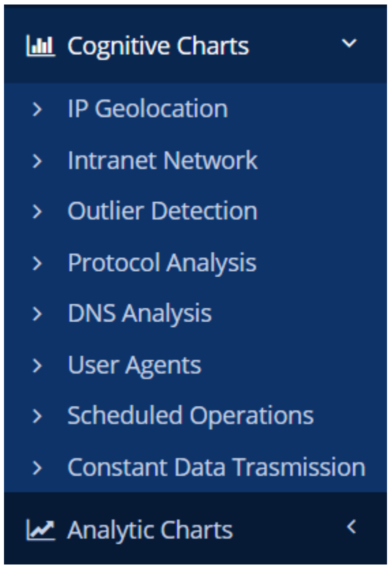


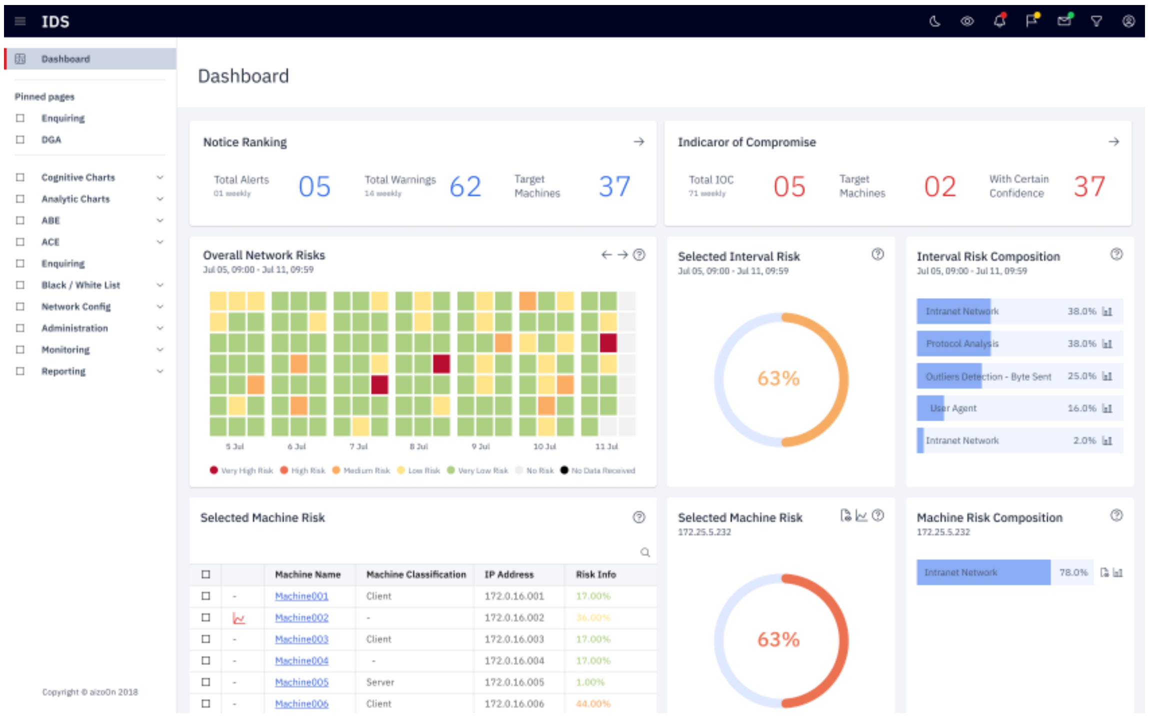
Figure 13: *Navigation menu*

Main navigation menu presents a minor issue with the use of icons: for options "Cognitive Charts" and "Analytic Charts" the arrow symbol is used to signify the presence of a dropdown menu, but for the options inside the dropdown is used as a bulleted list's bullet point. This inconsistency may lead to users invertedly navigating to the wrong page while trying to show more options. The effect of the violation of "Content 14: An icon should keep the same meaning in all the application" only has minor impact user experience and is easily revertible, so the severity of this problem is low.

5.4 Effect on IDS

The process of modernization of the front-end of IDS had among its objectives the solution of the interface problems that have just been presented; in this section the most relevant solutions will be analysed.

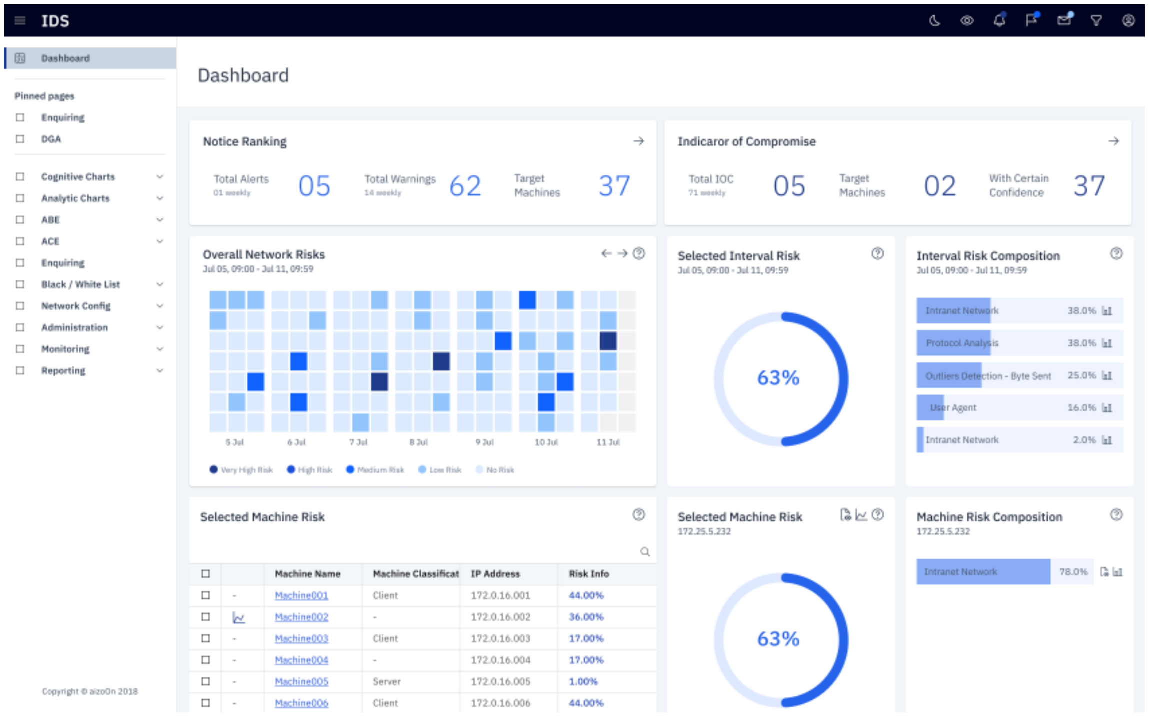
5.4.1 Dashboard



Selected Machine Risk

☐		Machine Name	Machine Classification	IP Address	Risk Info
☐	-	Machine001	Client	172.0.16.001	17.00%
☐		Machine002	-	172.0.16.002	36.00%
☐	-	Machine003	Client	172.0.16.003	17.00%
☐	-	Machine004	-	172.0.16.004	17.00%
☐	-	Machine005	Server	172.0.16.005	1.00%
☐	-	Machine006	Client	172.0.16.006	44.00%

Figure 14: *New dashboard, normal colours*



		Machine Name	Machine Classificat	IP Address	Risk Info
☐	-	Machine001	Client	172.0.16.001	44.00%
☐		Machine002	-	172.0.16.002	36.00%
☐	-	Machine003	Client	172.0.16.003	17.00%
☐	-	Machine004	-	172.0.16.004	17.00%
☐	-	Machine005	Server	172.0.16.005	1.00%
☐	-	Machine006	Client	172.0.16.006	44.00%

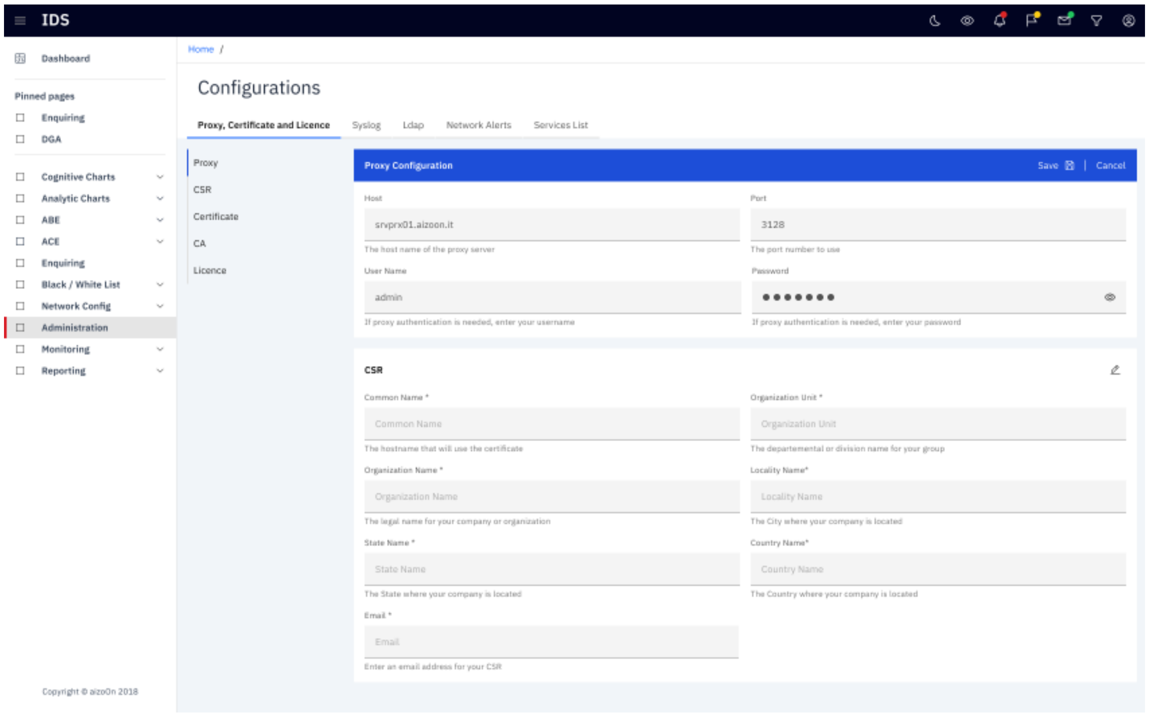
Figure 15: *New dashboard, colour-blindness support*

As it can be seen from *Figure 14* and *Figure 15* the dashboard (and all the pages containing graphs) now offers an accessible colour palette, that can be activated using the "eye" icon in the top of the page. Also, a dark theme is now available, that, even without bearing specific value in the sense of accessibility, can still be helpful in reducing users' eye strain.

Selected interval (and machine) risk levels and composition are now in separate containers, to better show the difference in meaning and both are now fully adaptive.

5.4.2 Configuration pages

The layout and task structure of configuration pages were unified to both create consistency and solve the interaction issues.



Configurations

Proxy Configuration

Host: srvprx01.aizoon.it
The host name of the proxy server

Port: 3128
The port number to use

User Name: admin
If proxy authentication is needed, enter your username

Password: •••••••
If proxy authentication is needed, enter your password

CSR

Common Name *: Common Name
The hostname that will use the certificate

Organization Unit *: Organization Unit
The departemental or division name for your group

Organization Name *: Organization Name
The legal name for your company or organization

Locality Name*: Locality Name
The City where your company is located

State Name *: State Name
The State where your company is located

Country Name*: Country Name
The Country where your company is located

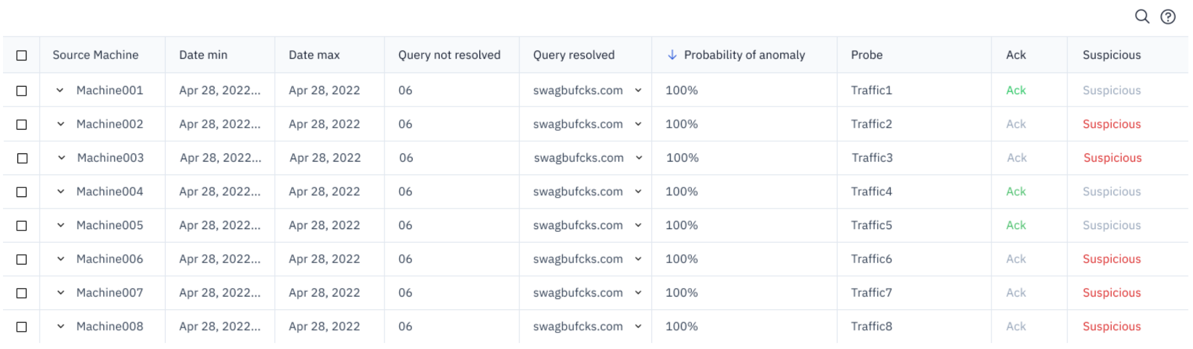
Email *: Email
Enter an email address for your CSR

Figure 16: *Configurations*

Now the editing can be enabled by clicking on the pencil icon, like the one in the top right angle of the "CSR" card, and the inserted data saved using the "save" button that appears in its place. Following guideline "Interactables 8: If some textbox has a specific requirement, it should be specified beforehand with a warning close to the textbox" details about each field have been specified directly under them.

5.4.3 Redundant table columns

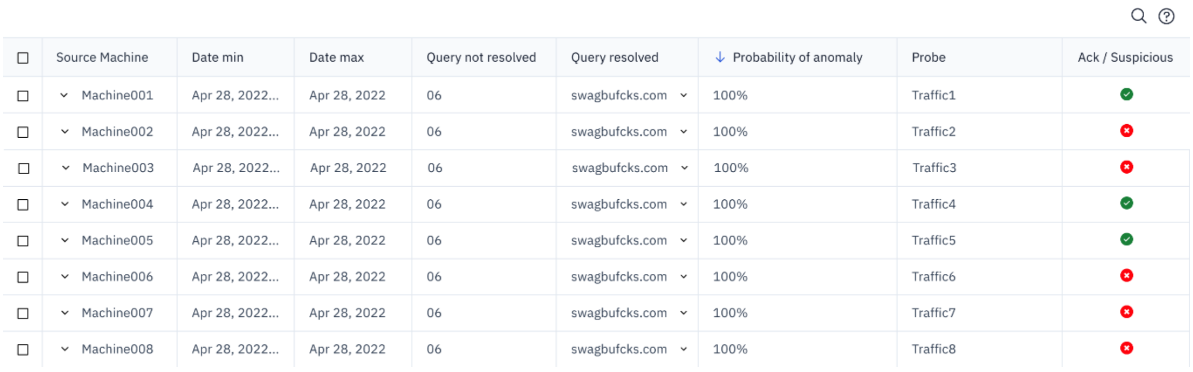
After the rearchitecting of IDS' front-end some problems originated in the new version of the interface: *Figure 17* shows a table with the risk associated with a series of machines, but the last two columns, "Ack" and "Suspicious", indicate some mutually exclusive information, going against guideline "Content 23: Table columns representing mutually exclusive information should be condensed into one". Furthermore, the way it's represented is not accessible to colour-blind people: for some types of colour blindness, especially for achromatopsia, the highlighted words look like the not highlighted ones, completely removing the ability of perceiving that information.



☐	Source Machine	Date min	Date max	Query not resolved	Query resolved	↓ Probability of anomaly	Probe	Ack	Suspicious
☐	⌄ Machine001	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic1	Ack	Suspicious
☐	⌄ Machine002	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic2	Ack	Suspicious
☐	⌄ Machine003	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic3	Ack	Suspicious
☐	⌄ Machine004	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic4	Ack	Suspicious
☐	⌄ Machine005	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic5	Ack	Suspicious
☐	⌄ Machine006	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic6	Ack	Suspicious
☐	⌄ Machine007	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic7	Ack	Suspicious
☐	⌄ Machine008	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic8	Ack	Suspicious

Figure 17: *Machine risk table with redundant columns (mock-up)*

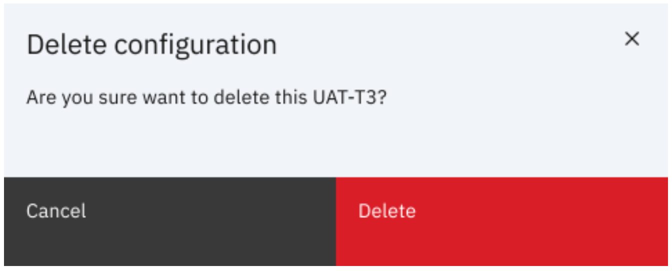
In *Figure 18* the proposed solution unites the two problematic columns into one and the use of different icons allow to convey the original meaning even in absence of colour.



☐	Source Machine	Date min	Date max	Query not resolved	Query resolved	↓ Probability of anomaly	Probe	Ack / Suspicious
☐	⌄ Machine001	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic1	✓
☐	⌄ Machine002	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic2	✗
☐	⌄ Machine003	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic3	✗
☐	⌄ Machine004	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic4	✓
☐	⌄ Machine005	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic5	✓
☐	⌄ Machine006	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic6	✗
☐	⌄ Machine007	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic7	✗
☐	⌄ Machine008	Apr 28, 2022...	Apr 28, 2022	06	swagbufcks.com ⌄	100%	Traffic8	✗

Figure 18: *Machine risk optimized table (mock-up)*

5.4.4 Confirmation dialog



Delete configuration

Are you sure want to delete this UAT-T3?

Figure 19: *Delete configuration dialog*

Despite not being an urgent problem, confirmation for all critical operations, like creation or deletion of configurations, has been improved, with clearer warnings and the use of appropriate colours. IDS' primary blue (RGB: 29,78,216) was used for confirmation operations and its error colour (RGB: 218, 30, 40) for deletion and dangerous operations.

5.5 Measures of success

During the interval of time taken in exam IDS was in active development, so, as a consequence of the stipulated guidelines, direct action has been taken.

In this period 24 total issues were identified, of which 18 directly linked to specific entries of the guidelines presented; 9 non-compliances were immediately fixed, 4 are being fixed in the forthcoming release and the remaining 11 are being worked on and will be of interest for future implementations.

Problems with the use of the security semaphore and colours chosen to represent it were recognised and have been object of further studies, especially in the other major applications by aizoOn, where colours used for product identity created confusion, and are being solved.

Following suggestion present in the guidelines check on contrast and colour difference have been implemented on IDS and other applications for both existing and future interface components.

Lastly pages handling sensible data, like administration ones, that provided bad user experience have been improved.

Overall, the interface guidelines where useful in displaying some problems that were being ignored and in having a clear unified reference on how to solve them, even if for a proper evaluation of their quality a higher amount of time and resources would be needed.

Improving the assortment of guidelines previously in use by adapting them to the specific context helped define new acceptance criteria, based on affordance, authority and familiarity. More over the definition of rules for accessibility like contrast and colour difference (delta E) thresholds made the application usable to a wider variety of people.

Chapter 6: Creating a workflow that prevents security problems

6.1 Motivation and objective of workflow guidelines

When developing an application, a lot of the practices and habits programmers follow reflect on the final product. Establishing rules for the development process that take security into consideration can prevent a number of issues that may arise later.

The following guidelines are meant to be interpreted as a brief supplement to the main ones developed in this thesis, highlighting few aspects that are often overlooked. They are not structured in a list of directives to follow, but as a series of considerations and suggestions to apply depending on the situation, specifying each time the reasoning behind it.

Especially in Agile development environments it’s easy to insert in the sprint the time dedicated to some good practices.

6.2 Workflow guidelines

6.2.1 Tools and frequency of use

To guarantee code security a wide assortment of tools is available to developers, however they are often underutilized or not properly integrated in the workflow.

Linters

Static analysis, in particular while the code is still being written can immediately identify future problems, even if sometimes warnings given are not about proper bugs, but about code smells that could become bugs and vulnerabilities at run time. Real-time linters should be always active when writing code and a complete code analysis should be performed at each commit, in order to correctly verify in which version a specific error has been introduced. SonarQube[18] and similar software offer this kind of functionalities and have integration for real-time linting with main IDEs.

This kind of tools have good scalability and the immediateness of use make them useful and efficient despite the fact that they can only identify a limited number of well-known vulnerabilities.

Testing

Testing is a key part of software development and effort should be spent in optimizing and performing the different types of testing that can be executed.

Unit testing should be performed each time a component is created or modified, to guarantee a basic level of correctness of everything that gets inserted in the code base, for application front-ends in JavaScript, like IDS, Jest is one of the most used tools to handle this task, also because of its compatibility with React. Testing should be security oriented, putting special attention on verifying the absence of potentially exploitable bugs (like data sources and sinks).

Integration testing is needed at least once per commit, as that is the moment in which potential issues may arise. If the back-end is properly tested, testing the front-end up until the API calls should be enough to guarantee total coverage.

Tools like Cypress[52] automate end-to-end and integration testing, by simulating user interaction, and should be used with the same frequency of integration testing, as they cover the same role, but with a parallel approach.

Keeping track of the test results from one commit to another can greatly reduce the effort needed to identify and subsequently fix bugs.

Implementing accessibility testing in the development process is strongly suggested, products like Axe[53] or Wave[54] offer dedicated functionalities to test the accessibility of a web applications, but also automated end-to-end testing tools like the previously mentioned Cypress give the possibility to write tests that verify colours of items and other accessibility requirements indirectly. Integration of AXE with Cypress is possible thanks to a dedicated component called cypress-axe[55], that confronts pages against a series of customizable rules.

6.2.2 Defining custom guidelines

To be successful a system has to take into account the exceptions that might occur and the modification and adaptation of itself to handle those exceptions. Both the main guidelines presented and the content of the present supplement can and should be manipulated to better fit the specific situation, this section proposers a simple structured way to do it.

When creating a web application, it’s common practice to adopt an existing set of guidelines, however every product is different and a variety of reason can bring developers to break the chosen rules or to define new ones. Customizing someone’s own guidelines has a lot of benefits, but the process itself need to follow a precise method:

- Each new rule added should be strictly necessary, if an exception can already be envisioned, then the rule should be formulated in a way that includes that exemption.

- Every time an existing rule is changed the reason for the change should be known and explicitly stated in an appropriate document.
- Complete removal of rules should be limited to rules that are not appliable to the system they should refer to, in other cases adapting the rule is always better.

Changing a set of guidelines following these three simple rules guarantees a formalized output that does not deviate from the original scope and keeping track of the changes is useful to transmit the knowledge and the though process to future users.

6.2.3 Documentation and knowledge base

In a development studio the programming base can change a lot through time, the obvious problem is that a huge amount of knowledge about the application, especially the thought process behind decisions, is not directly extractable from the code. Misunderstandings on a component's intended function can cause significant design errors, that will eventually impact security. Ins erting tasks about writing proper documentation into the agile sprint means saving resources each time a new employee is hired; a programmer, even with limited experience can easily understand anything they need from a good documentation without having someone else invest time in explaining it.

If an existing project has no documentation, slowly making one, starting from components in active development, is still better than expanding the amount of non-documented code.

6.2.4 Structuring discovery

New tools and technologies related to web application are published every day, consequently keeping up with the most recent changes is imperative in order to provide the best possible product, even from a security standpoint. People with a deep technical knowledge are the most suited to discover new tools to use in the project, so including even a few hours dedicated to discovering and trying out new technologies into each sprint can increase the quality of the final result and create opportunities for improvement.

6.3 Measures of success

Creating a unified document containing all the guidelines needed for development, for both the product and the process, and focused on the security impact individual decision can have helped developers and designers streamline the internal mechanisms.

A more widespread awareness of additional criteria to consider during development has been observed in the IDS development team.

Chapter 7: Conclusions and future work

7.1 Key findings and contribution to the field

The present exploration meant to present an under-researched aspect of interface usability, namely the impact it can have on the security of safety critical applications. Small graphical changes can affect human computer interactions to the point of compromising security, by accidentally incentivising dangerous behaviours.

The context of an application under the process of modernization granted the opportunity of testing the developed guideline, main objective of the thesis, on a dynamic environment and to witness its evolution in real time.

Commonly accepted guidelines collected by the initial literature review were applied and tested to verify the extent of their effectiveness and effect on security, then the developed interface guideline underwent a similar but more extensive procedure and were applied to the preexisting interface.

Simplifying the corpus of rules to follow was successful in optimizing the overall workflow of designers and developers and improved their awareness for themes like accessibility and usability, in their specificity for safety critical applications.

7.2 Limitations and future research directions

Modernization of applications concerns a wide variety of existing cases, so it is of high priority for organization developing web applications to integrate continuous update of the system as a normal part of development and a prevention strategy, too often modernization is considered a once-in-a-while practice just to solve unavoidable and manifest problems.

When modernization concerns user experience the complexity of the situations increases, as it stops being simply a code related issue and starts involving the relationship between humans and machines.

Both theoretical research and concrete application executed for this thesis suffered from the limited resources and time available and would benefit from additional in-depth work on specific sections. The topic offers a number of cues for research from both a technical and humanistic point of view. Further technical research needs to be done to develop better tools to guarantee more effective, and possibly inclusive, designs and psychological studies dedicated on influence and interpretation of information should be expanded and brought to the attention of developers and designers.

In both cases the focus is on human factors and on how to preemptively avoid the problems they can cause instead of trying to handle them after they appear.

It’s important to remember that every technical issue that involves humans is first of all a human issue.

References

[1] "Application Development Software Market Size, Trends and Global Forecast To 2032." Accessed: Aug. 29, 2023. [Online]. Available: https://www.thebusinessresearchcompany.com/report/application-development-software-global-market-report

[2] "What is web application security? | Web security," Cloudflare. Accessed: Aug. 25, 2023. [Online]. Available: https://www.cloudflare.com/learning/security/what-is-web-application-security/

[3] Archiveddocs, "Web Application Security Fundamentals." Accessed: Aug. 25, 2023. [Online]. Available: https://learn.microsoft.com/en-us/previous-versions/msp-n-p/ff648636(v=pandp.10)

[4] "OWASP Top 10:2021." Accessed: Aug. 22, 2023. [Online]. Available: https://owasp.org/Top10/

[5] "Gartner Top Security and Risk Trends in 2022," Gartner. Accessed: Jul. 11, 2023. [Online]. Available: https://www.gartner.com/en/articles/7-top-trends-in-cybersecurity-for-2022

[6] "Gartner Forecasts 39% of Global Knowledge Workers Will Work Hybrid by the End of 2023," Gartner. Accessed: Jul. 12, 2023. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2023-03-01-gartner-forecasts-39-percent-of-global-knowledge-workers-will-work-hybrid-by-the-end-of-2023

[7] "Gartner Survey Shows 75% of Organizations Are Pursuing Security Vendor Consolidation in 2022," Gartner. Accessed: Aug. 16, 2023. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-09-12-gartner-survey-shows-seventy-five-percent-of-organizations-are-pursuing-security-vendor-consolidation-in-2022

[8] "2022," Verizon Business. Accessed: Aug. 25, 2023. [Online]. Available: https://www.verizon.com/business/en-gb/resources/reports/dbir/2022/

[9] "What Is a Legacy System and What Are Legacy Applications? | Definition from TechTarget.com," IT Operations. Accessed: Aug. 29, 2023. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/legacy-application

[10] "App modernization 101: Understand your options—and how to get started | TechBeacon." Accessed: Aug. 29, 2023. [Online]. Available: https://techbeacon.com/app-dev-testing/app-modernization-101-understand-your-options-how-get-started

[11] "7 Options To Modernize Legacy Systems." Accessed: Aug. 29, 2023. [Online]. Available: https://www.gartner.com/smarterwithgartner/7-options-to-modernize-legacy-systems

[12]“CWE - CWE-1333: Inefficient Regular Expression Complexity (4.12).” Accessed: Aug. 02, 2023. [Online]. Available: https://cwe.mitre.org/data/definitions/1333.html

[13]“Catastrophic backtracking.” Accessed: Aug. 02, 2023. [Online]. Available: https://javascript.info/regexp-catastrophic-backtracking

[14]“CVE-2022-25883 - GitHub Advisory Database,” GitHub. Accessed: Aug. 02, 2023. [Online]. Available: https://github.com/advisories/GHSA-c2qf-rxjj-qqgw

[15]“CVE-2023-26115 - GitHub Advisory Database,” GitHub. Accessed: Aug. 02, 2023. [Online]. Available: https://github.com/advisories/GHSA-j8xg-fqg3-53r7

[16] “GHSA-36jr-mh4h-2g58 - GitHub Advisory Database,” GitHub. Accessed: Aug. 02, 2023. [Online]. Available: https://github.com/advisories/GHSA-36jr-mh4h-2g58

[17]“Grit Documentation.” Accessed: Aug. 18, 2023. [Online]. Available: https://docs.grit.io/

[18] “Code Quality Tool & Secure Analysis with SonarQube.” Accessed: Aug. 31, 2023. [Online]. Available: https://www.sonarsource.com/products/sonarqube/

[19] “10 Front-End Security Best Practices,” CLIMB. Accessed: Aug. 25, 2023. [Online]. Available: https://climbtheladder.com/10-front-end-security-best-practices/

[20] “Front-End Architecture Best Practices - OutSystems Best Practices.” Accessed: Aug. 25, 2023. [Online]. Available: https://success.outsystems.com/documentation/best_practices/development/front_end_architecture_best_practices/

[21]“5 Best Practices For Optimizing Your Front End Web Development.” Accessed: Aug. 25, 2023. [Online]. Available: https://pxmatrix.com/best-practices-for-optimizing-your-front-end-web-development/

[22] B. Shneiderman, C. Plaisant, M. Cohen, S. Jacobs, N. Elmqvist, and N. Diakopoulos, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 6th ed. Pearson, 2016.

[23] “Deceptive Patterns - Home.” Accessed: Aug. 28, 2023. [Online]. Available: https://www.deceptive.design/

[24] J. Benway and D. M. Lane, “Banner Blindness: Web Searchers Often Miss ‘Obvious’ Links,” 1998. Accessed: Aug. 28, 2023. [Online]. Available: https://www.semanticscholar.org/paper/Banner-Blindness%3A-Web-Searchers-Often-Miss-Links-Benway-Lane/2e153327467dc3825fb963e9d0c92193ab5983bd

[25] W. L. in R.-B. U. Experience, “Security & Human Factors,” Nielsen Norman Group. Accessed: Aug. 26, 2023. [Online]. Available: https://www.nngroup.com/articles/security-and-human-factors/

[26] W. L. in R.-B. U. Experience, “User Education Is Not the Answer to Security Problems,” Nielsen Norman Group. Accessed: Aug. 26, 2023. [Online]. Available: https://www.nngroup.com/articles/security-and-user-education/

[27] S. Gupta and S. Furnell, “From Cybersecurity Hygiene to Cyber Well-Being,” in *HCI for Cybersecurity, Privacy and Trust*, A. Moallem, Ed., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 124–134. doi: 10.1007/978-3-031-05563-8_9.

[28] D. A. Norman, *The design of everyday things*, 1st Basic paperback. New York: Basic Books, 2002.

[29] J. J. Gibson, *The Ecological Approach to Visual Perception: Classic Edition*. Psychology Press, 2014.

[30] “Umberto Eco: cari filosofi è l’ora del Realismo Negativo - la Repubblica.it,” Archivio - la Repubblica.it. Accessed: Aug. 26, 2023. [Online]. Available: https://ricerca.repubblica.it/repubblica/archivio/repubblica/2012/03/11/umberto-eco-cari-filosofi-ora-del-realismo.html

[31]W. L. in R.-B. U. Experience, “The Authority Principle,” Nielsen Norman Group. Accessed: Aug. 26, 2023. [Online]. Available: https://www.nngroup.com/articles/authority-principle/

[32] “Milgram experiment | Description, Psychology, Procedure, Findings, Flaws, & Facts | Britannica.” Accessed: Aug. 28, 2023. [Online]. Available: https://www.britannica.com/science/Milgram-experiment

[33] “Variations on Practiced Patterns Cause Mistakes.” Accessed: Aug. 28, 2023. [Online]. Available: https://www.nngroup.com/articles/practiced-patterns-mistakes/

[34] J. Yablonski, “Jakob’s Law,” Laws of UX. Accessed: Aug. 28, 2023. [Online]. Available: https://lawsofux.com/jakobs-law/

[35] D. L. Everett, “Cultural Constraints on Grammar and Cognition in Pirahã: Another Look at the Design Features of Human Language,” *Curr. Anthropol.*, vol. 46, no. 4, pp. 621–646, Aug. 2005, doi: 10.1086/431525.

[36] B. M. Sobel and V. K. Sims, “Anthropomorphic Perceptions of Simple Text-Based Interfaces,” in *Human-Computer Interaction. User Experience and Behavior*, M. Kurosu, Ed., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 233–242. doi: 10.1007/978-3-031-05412-9_17.

[37] “Download the 2021 Insider Data Breach Survey Report | Egress.” Accessed: Aug. 31, 2023. [Online]. Available: https://www.egress.com/blog/what-is-human-layer-security/2021-insider-breach-survey

[38] “Designing UIs with Sense of Security in mind | by Maria Margarida | UX Collective.” Accessed: Jul. 11, 2023. [Online]. Available: https://uxdesign.cc/designing-uis-with-security-in-mind-f4f6f265573a

[39] W. W. A. Initiative (WAI), “Cognitive Accessibility at W3C,” Web Accessibility Initiative (WAI). Accessed: Aug. 28, 2023. [Online]. Available: https://www.w3.org/WAI/cognitive/

[40] A. Alam, R. Biddle, and E. Stobert, “Emics and Etics of Usable Security: Culturally-Specific or Culturally-Universal?,” in *HCI for Cybersecurity, Privacy and Trust*, A. Moallem, Ed., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 22–40. doi: 10.1007/978-3-030-77392-2_2.

[41]M. Hertzum, “Wellbeing at Work: Four Perspectives on What User Experiences with Artifacts May Contribute,” in *Beyond Interactions*, J. Abdelnour Nocera, A. Parmaxi, M. Winckler, F. Loizides, C. Ardito, G. Bhutkar, and P. Dannenmann, Eds., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 19–25. doi: 10.1007/978-3-030-46540-7_2.

[42] J. Kammersgaard, “Four different perspectives on human–computer interaction,” *Int. J. Man-Mach. Stud.*, vol. 28, no. 4, pp. 343–362, Apr. 1988, doi: 10.1016/S0020-7373(88)80017-8.

[43] D. Jacobs and T. McDaniel, “A Survey of User Experience in Usable Security and Privacy Research,” in *HCI for Cybersecurity, Privacy and Trust*, A. Moallem, Ed., in Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 154–172. doi: 10.1007/978-3-031-05563-8_11.

[44] G. Sharma, W. Wu, and E. N. Dalal, “The CIEDE2000 color-difference formula: Implementation notes, supplementary test data, and mathematical observations,” *Color Res. Appl.*, vol. 30, no. 1, pp. 21–30, Feb. 2005, doi: 10.1002/col.20070.

[45] “CIE | International Commission on Illumination / Comission internationale de l’Eclairage / Internationale Beleuchtungskommission.” Accessed: Sep. 25, 2023. [Online]. Available: http://cie.co.at/

[46] “Web Content Accessibility Guidelines (WCAG) 2.1.” Accessed: Sep. 25, 2023. [Online]. Available: https://www.w3.org/TR/2023/REC-WCAG21-20230921/#non-text-contrast

[47] W. L. in R.-B. U. Experience, “F-Shaped Pattern of Reading on the Web: Misunderstood, But Still Relevant (Even on Mobile),” Nielsen Norman Group. Accessed: Aug. 29, 2023. [Online]. Available: https://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/

[48] W. L. in R.-B. U. Experience, “The Layer-Cake Pattern of Scanning Content on the Web,” Nielsen Norman Group. Accessed: Aug. 29, 2023. [Online]. Available: https://www.nngroup.com/articles/layer-cake-pattern-scanning/

[49] W. L. in R.-B. U. Experience, “Interface Copy Impacts Decision Making,” Nielsen Norman Group. Accessed: Aug. 29, 2023. [Online]. Available: https://www.nngroup.com/articles/interface-copy-decision-making/

[50] “Five Interaction Styles - Computing and Software Wiki.” Accessed: Aug. 29, 2023. [Online]. Available: http://wiki.cas.mcmaster.ca/index.php/Five_Interaction_Styles

[51]A. A. H. J. Thiadens *et al.*, “Genetic Etiology and Clinical Consequences of Complete and Incomplete Achromatopsia,” *Ophthalmology*, vol. 116, no. 10, pp. 1984-1989.e1, Oct. 2009, doi: 10.1016/j.ophtha.2009.03.053.

[52] “JavaScript Component Testing and E2E Testing Framework | Cypress.” Accessed: Aug. 31, 2023. [Online]. Available: https://www.cypress.io/

[53] “axe: Accessibility Testing Tools and Software,” Deque. Accessed: Aug. 31, 2023. [Online]. Available: https://www.deque.com/axe/

[54] “WAVE Web Accessibility Evaluation Tools.” Accessed: Aug. 31, 2023. [Online]. Available: https://wave.webaim.org/

[55] “cypress-axe.” component-driven, Aug. 16, 2023. Accessed: Aug. 31, 2023. [Online]. Available: https://github.com/component-driven/cypress-axe

Appendix A

Before the fix

These, ordered in tables, are the outputs of the commands executed on the code before any change was performed on it:

Number	Severity	Details	Package	Patched in	Dependency of	Path	More info
1	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=5.7.2	patch-package	patch-package > semver	https://www.npmjs.com/advisories/1092459
2	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=5.7.2	patch-package	patch-package > cross-spawn > semver	https://www.npmjs.com/advisories/1092459
3	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=5.7.2	stylelint	stylelint > meow > read-pkg-up > read-pkg > normalize-package-data > semver	https://www.npmjs.com/advisories/1092459
4	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=6.3.1	eslint-config-airbnb-base	eslint-config-airbnb-base > semver	https://www.npmjs.com/advisories/1092460

5	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=6.3.1	eslint-config-airbnb	eslint-config-airbnb > eslint-config-airbnb-base > semver	https://www.npmjs.com/advisories/1092460
6	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=6.3.1	c8	c8 > istanbul-lib-report > make-dir > semver	https://www.npmjs.com/advisories/1092460
7	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=6.3.1	c8	c8 > istanbul-reports > istanbul-lib-report > make-dir > semver	https://www.npmjs.com/advisories/1092460
8	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=6.3.1	vite-plugin-svgr	vite-plugin-svgr > @svgr/core > @svgr/plugin-jsx > @babel/core > @babel/helper-compilation-targets > semver	https://www.npmjs.com/advisories/1092460
9	moderate	semver vulnerable to Regular Expression Denial of Service	semver	>=6.3.1	vite-plugin-pwa	vite-plugin-pwa > workbox-build > @babel/preset-env > babel-plugin-polyfill-corejs2 > @babel/helper-define-polyfill-provider > @babel/helper-compilation-targets > semver	https://www.npmjs.com/advisories/1092460
10	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/maps	@ant-design/maps > @antv/l7 > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594
11	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/maps	@ant-design/maps > @antv/l7 > @antv/l7-core > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594
12	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/maps	@ant-design/maps > @antv/l7 > @antv/l7-component > @antv/l7-core > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594

13	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @ant-design/maps > @antv/l7 > @antv/l7-component > @antv/l7-core > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594
14	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @ant-design/maps > @antv/l7 > @antv/l7-scene > @antv/l7-component > @antv/l7-core > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594
15	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @ant-design/maps > @antv/l7 > @antv/l7-scene > @antv/l7-layers > @antv/l7-maps > @antv/l7-core > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594
16	moderate	word-wrap vulnerable to Regular Expression Denial of Service	word-wrap	>=1.2.4	eslint	eslint > optionator > word-wrap	https://www.npmjs.com/advisories/1092593
17	moderate	word-wrap vulnerable to Regular Expression Denial of Service	word-wrap	>=1.2.4	jsdom	jsdom > escodegen > optionator > word-wrap	https://www.npmjs.com/advisories/1092593

After the fix

Number	Severity	Details	Package	Patched in	Dependency of	Path	More info
1	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @antv/g2plot > @antv/g-base > d3-interpolate > d3-color	https://www.npmjs.com/advisories/1088594
2	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @antv/g2plot > @antv/g2 >	https://www.npmjs.com/advisories/1088594

						@antv/g-base > d3-interpolate > d3-color	
3	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @antv/g2plot > @antv/g2 > @antv/component > @antv/g-base > d3-interpolate > d3-color	https://www.npmjs.com/advisories/1088594
4	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/charts	@ant-design/charts > @antv/g6 > @antv/g6-pc > @antv/g6-plugin > @antv/g-canvas > @antv/g-base > d3-interpolate > d3-color	https://www.npmjs.com/advisories/1088594
5	high	d3-color vulnerable to ReDoS	d3-color	>=3.1.0	@ant-design/maps	@ant-design/maps > @antv/l7 > @antv/l7-utils > d3-color	https://www.npmjs.com/advisories/1088594