Hack in an Elevator! Pentesting a Lift Control Web App

Pericle Perazzo
Department of Information Engineering (DII)
University of Pisa
Pisa, Italy, EU
pericle.perazzo@unipi.it

Gianmarco Manfredonia
Department of Information Engineering (DII)
University of Pisa
Pisa, Italy, EU
g.manfredonia2@studenti.unipi.it

Abstract

Imagine taking an elevator to go to the fourth floor, and suddenly you are stuck inside due to a cyber attack! This can happen since elevators have become Cyber-Physical Systems (CPS), which involve networked embedded computers, and therefore they are not anymore immune to hackers. In this research we assess the security of an elevator CPS designed and developed by an Italian company, which is deployed on several elevator installations in Italy. The objective is to evaluate if and to what extent the various cybersecurity risks are understood by CPS developers. In this paper we present the results of the first part of a complete penetration test, in which we focused on the elevator management web site only, which is the component most exposed to possible attacks due to its public and remote availability. From our experience we can conclude that the CPS developers have a good awareness of the most common cybersecurity threats, and they are aware of common defense techniques. Still, they miss to implement defenses against advanced client-side attacks, and they do not follow the best practices, which could lead to vulnerabilities in case some unfortunate conditions are met.

CCS Concepts

• **Security and privacy** → **Web application security**; • **Computer systems organization** → *Embedded systems.*

Keywords

Cybersecurity, Cyber-Physical Systems, Penetration Test, Web Security

ACM Reference Format:
Pericle Perazzo and Gianmarco Manfredonia. 2018. Hack in an Elevator! Pentesting a Lift Control Web App. In *Proceedings of 21st International Conference on Embedded Wireless Systems and Networks (EWSN '24).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Cyber-Physical Systems (*CPS*) represent a new generation of systems that integrate computational, control and physical components. CPSes are at the heart of modern industries and critical infrastructures such as power grids, transportation systems, and healthcare systems. Unfortunately, the integration of computation, network, and physical components creates big complexity and expands the attack surface of the system, making cybersecurity a significant concern. The problem of cybersecurity in CPSes is critical, because a security breach could lead to catastrophic consequences, and security solutions also need to consider aspects such as safety, reliability, and real-time constraints. It is paramount that CPSes are designed and programmed with security in mind. However, the general cybersecurity awareness of the nowadays average CPS developer is still unclear. Some reasearch in this direction exists, but it focuses mainly in industrial CPSes or critical infrastructures. Less research has been devoted to the CPSes that we use in our everyday life, like those employed in residential or office buildings.

In this research we assess the security of an elevator CPS designed and developed by an Italian company, which is deployed on non-industrial buildings (homes, offices, etc.) in Italy, by performing a white-box penetration test. The objective is to evaluate if and to what extent the various cybersecurity risks are understood by real non-industrial CPS developers. An elevator CPS is composed of both physical components (like the elevator car, the electric motor that moves the car, the physical buttons inside the elevator, etc.) and computation components (like the embedded computer boards that controls the movement of the elevator, their firmware, the software for remote monitoring and management, etc.).

In this paper we present the results of the first part of the penetration test, in which we focused on the elevator management web site, which is the component most exposed to possible attacks due to its public and remote availability. The results of the penetration test on the other CPS components, for example the controller board firmware, is left for a future paper. From our experience we can conclude that, for what concerns the web part, the CPS developers have a good awareness of the most common cybersecurity threats, and they are aware of common defense techniques like anti-CSRF tokens [3]. Still, although we did not find any major and easily exploitable vulnerability, some defenses against advanced client-side attacks like clickjacking [2] are missing. Developers adopted some defense techniques without following the best practices (e.g., they put secrets in URLs), and without being aware of some tricky server-side language functionalities (e.g., PHP type juggling [15]). They also do not employ some free defense-in-depth mechanisms, which could give more security without any impact on performance or complexity. This negligence could lead to vulnerabilities in future versions of the site, or in case some unfortunate conditions are met.

The rest of the paper is structured as follows. Section 3 introduces some preliminary concepts about the attacks that we will assess in the following sections. Section 4 describes the elevator CPS in general and its management web site in particular. Sections 5, 6, and 7 report our experiences in pentesting the web site respectively for what concerns authentication and SQL vulnerabilities, cookie

EWSN '24, December 10–13, 2024, Abu Dhabi, UAE
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

and clickjacking vulnerabilities, and vulnerabilities related to the anti-CSRF token. Section 2 reports some related work. Finally, the paper is concluded in Section 8.

2 Related Work

After the W32.Stuxnet incident [7], the interest on the security of CPSes has grown considerably [5]. Scientific research about CPS security broadly divide in two main topics: CPS-oriented intrusion detection systems [11, 13, 14, 20], and CPS security assessments [4, 16, 18]. Of course, the present paper fits better with the CPS security assessment track, so in the following we will compare with such works exclusively.

Apa [4] and, independently, Quarta et al. [18] carried out extensive penetration tests on industrial robots. Apa [4] identified almost 50 crucial security vulnerabilities in top-tier collaborative robots such as Baxter/Sawyer from Rethink Robotics and UR from Universal Robots. Quarta et al. [18] conducted a thorough investigation of the typical structure of an industrial robot and analyzed a specific implementation (namely the ABB 6-axis IRB140 robot) from a system security perspective. The authors modeled a potential attacker which considers the fundamental requirements of accurate environmental sensing, correct control logic execution, and operator safety. Quarta et al. also focused on CPS-specific attacks like control loop alteration and calibration parameters tampering.

Pogliani et al. [16] give a comprehensive view of the security issues that arise in designing and securely deploying controlled manufacturing systems. The authors also suggest practical measures to safeguard industrial cyber-physical systems and discuss the inadequacies of existing standards in industrial robotics to address active threats. Notably, past research about CPS security assessment almost exclusively focused on industrial CPSes (smart factories, connected robots, etc.) and critical infrastructure CPSes (smart grids, power plants, healthcare systems, etc.). Little research has been devoted to “civilian” CPSes like elevators installed in residential buildings and offices. In this paper we assess the security of an elevator CPS deployed on non-industrial buildings (homes, offices, etc.).

3 Preliminaries

In the following, we give some preliminary concepts about the attacks that we will assess in Sections 5, 6, and 7.

3.1 PHP Type Juggling

PHP is a loosely typed programming language, which means that when the programmer defines a variable, he does not need to declare a type for it. Internally, PHP will determine the type of a variable by the context in which such a variable is used. *Type juggling* in PHP refers to the automatic conversion of variables from one type to another [15]. This happens implicitly in certain situations without needing an explicit conversion. PHP performs type juggling for example during the comparison of variables of different types and in arithmetic operations. For instance, the expression `1=="1abc"` will return `true`, because PHP converts the right operand to the integer 1, in such a way to execute the comparison between variables that have the same type.

While type juggling can be convenient in some cases, it can also lead to unexpected results if not handled carefully [10]. Typically, vulnerabilities can arise in case type juggling is combined with a deserialization flaw. For example if the application accepts input via `json_decode()`, which can produce JSON objects with non-string fields, without checking the type of such fields. A common example is a broken authentication vulnerability, in which the PHP code looks like this: `if ($input == "Admin_Password") {login_as_admin();}`, simply submitting the integer input `0` would successfully bypass the password check, since the expression `(0 == "Admin_Password")` will evaluate to `true`. To prevent type juggling vulnerabilities it is recommended to always check the type of the inputs to be the expected one, and to replace loose comparisons (`==`, `!=`) with strict comparisons (`===`, `!==`), which check the operands' types and values to be equal.

3.2 Cross-Site Request Forgery and Anti-CSRF Tokens

Cross-Site Request Forgery (*CSRF*) is a client-side attack that tricks an authenticated user into executing unwanted actions on a web application [8]. The victim is typically induced to click a malicious link, or visit a malicious third-party site with an auto-submitting form, etc. The actions involuntary performed by the victim could include changing a password, making a purchase, or any other state-changing request that is typically (but not exclusively) performed via POST HTTP requests. Note that the session token cannot defend against CSRF, because it is usually implemented as a cookie (e.g., the customary `PHPSESSID` cookie of PHP) and sent by the browser along with every request, including the malicious CSRF ones.

On the other hand, *anti-CSRF tokens* are designed to specifically prevent CSRF [3]. An anti-CSRF token is a unique and unpredictably random code generated by the server and inserted into HTML forms, typically as a hidden field. The browser mirrors back the anti-CSRF token through the request body when submitting the form, and the server must check its presence and correctness before performing the action. This mechanism effectively prevents CSRF attacks, as long as the attacker cannot steal or guess the anti-CSRF token itself.

3.3 Clickjacking

Clickjacking is an interface-based client-side attack where a user is tricked into clicking on actionable content of a honest web site by clicking on some other content of a malicious decoy web site [2]. Typically, an attacker incorporates the target site as an iframe layer overlaid on the decoy site. The iframe is positioned within the user interface in such a way that there is a precise overlap between the actionable content of the honest site and an innocuous-looking content of the decoy site. When a user interacts with what he believes to be the decoy website, he is actually interacting with the hidden target site, performing unwanted actions on it. Note that neither the session token nor the anti-CSRF token can defend against clickjacking, because the user session is established with content loaded from an authentic (though framed) site and with all requests happening on-site. Therefore, both session and anti-CSRF tokens are included into requests and passed to the server as part of a normally behaved session.

To prevent clickjacking vulnerabilities, it is recommended to use Content Security Policy (CSP), and in particular the `frame-ancestors` CSP directive with a value of `none` for all the returned pages. In this way, the client will be prevented from embedding any page inside an iframe or similar HTML elements.

4 System Description and Design

The target of our penetration test is a CPS composed of a series of elevators installed on various buildings of different customers. Each elevator is equipped with a series of embedded computer boards: two of them are inside the elevator car, one is at the bottom of the shaft, and one is at each floor's door. One of these boards provides Internet connectivity for the others, via authenticated Wi-Fi. The boards periodically download commands from a management web site, through an authenticated connection. The elevator management site is intended to be used by maintenance technicians equipped with appropriate credentials. Through this site, the technician can control installation parameters, check the status of each elevator, and consult a log containing errors and activities.

In this first part of the penetration test we focused on the elevator management site only, because it is the component most exposed to possible attacks due to its public and remote availability. Testing the other components, for example the embedded computers, their firmware and their Wi-Fi connectivity, is left for a future work. The experiences reported in the present paper focus exclusively on the elevator management site penetration test.

The web site is written in the HTML and JavaScript languages, and it uses PHP for the back-end logic. It also uses a MySQL database for storing credentials, elevators information, etc. Figure 1 shows the site page transition scheme. Before doing anything the

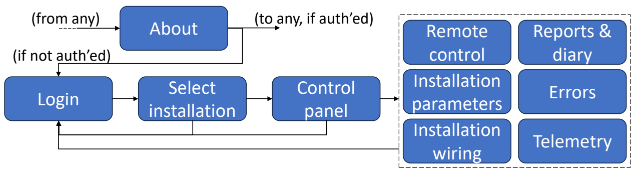


Figure 1: Page transition scheme of the elevator management web site

user has to log in on the "Log in" page with a username and a password, after which the flow will be redirected to the "Select installation" page, which make the user select an elevator that he wants to monitor and manage. After choosing the elevator, the "Control panel" page will be loaded with information regarding the chosen elevator. From this page, the user can freely move between various other pages ("Remote control", "Installation parameters", etc.), thanks to a menu. The most critical of these pages is the "Remote control" one, through which the user can close and open the elevator car's door, move the elevator, or make it stop at any moment. The user cannot select another elevator unless he logs out. The logout action can be performed at any page except "Login", and it redirects the user to the "Login" page. The "About" page is always accessible, by authenticated as well as anonymous users.

5 Authentication and SQL Injection Vulnerabilities

The elevator management site authenticates the user by means of username and password information, which the user provides as request body parameters. The back-end script then takes these parameters and checks for the validity of such credentials by querying the MySQL database. In particular, it performs a `SELECT` query on a credential table, with a `WHERE` clause that selects the row with the specified username. If the query returns at least a row, then a hash of the password is retrieved from such a row, and the password provided by the user is verified against it. If this final check succeeds, then the authentication is successful. The web site uses the bcrypt randomized hash algorithm for the passwords, to avoid possible rainbow attacks against the database content [17].

The MySQL query is not carried out by means of prepared statements, which is the most safe and recommended technique to avoid SQL injection attacks [1, 9]. However, the username input is sanitized with the `mysqli_real_escape_string()` function, which escapes characters that can be interpreted as string delimiters or can cause the legitimate string delimiters to be ignored, namely single quotes (`'`), double quotes (`"`), and backslashes (`\`). Consequently, the login functionality turned out to be immune to SQL injection attacks. SQL queries are used in other parts of the site as well, but the back-end scripts extensively use a series of escaping functions on inputs before placing them into any query. In particular, string inputs which are *not* employed in `LIKE` operations are escaped by the `mysqli_real_escape_string()` function. Those which are employed in `LIKE` operations are escaped on two additional characters, namely percentages (`%`) and underscores (`_`), thanks to the `addcslashes()` function. Finally, the string inputs which are used as names of tables and columns are sanitized by stripping dangerous characters selected by a regular expression, thanks to the `preg_replace()` function. The employed regular expression matches any character that is not a lowercase or uppercase letter, a number, or an underscore, and it removes it from the input.

We conducted a series of injection probes with the SQLMap tool, launched with various configurations. We also tried to mount well-known SQL injection techniques, such as piggybacked queries, tautologies, union operators, error-based SQL injections, and blind SQL injections. From the provided tests and code analysis, no evident SQL injection vulnerability emerged from any part of the site. Nevertheless, developers should use prepared statements [9] instead of escaping in SQL queries whenever possible. Escaping defenses are sometimes error-prone and difficult to apply correctly, and they could produce severe vulnerabilities in future versions of the site.

Coming back to the login functionality, we observed that there is no control over the number of login attempts made, neither taking into account the IP address they come from nor the username that is trying to log in. This leaves room for brute-force attacks using publicly available wordlists of common passwords [12]. The attacker could also use a wordlist specifically crafted for the elevator management site, by retrieving and combining various information regarding the company and its employees, obtained by social engineering. The web site does not have a user registration functionality, and new users can be registered only by the site administrator, by

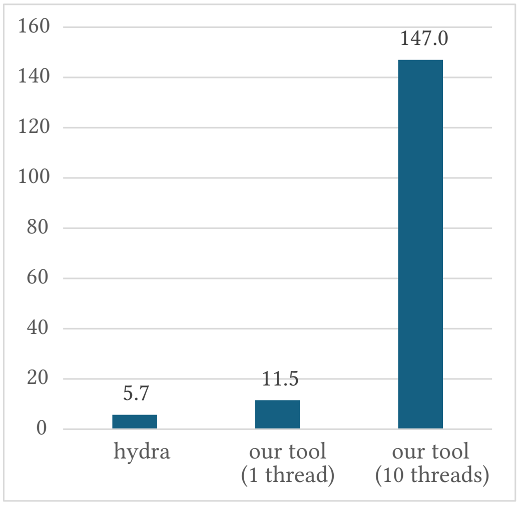


Figure 2: Login brute-force throughput

interacting directly with the MySQL server and manually adding tuples into the credentials table. Therefore, the actual strength of the authentication mechanism relies on the users' capacity of creating strong passwords, or the administrator's capacity of creating strong passwords for them. Of course, this policy scales badly in case the number of users increases, so the site should be equipped with a user registration functionality, employing an automated password strength estimation.

We noted that the webspace provider employs a mild protection against login brute-force attacks (and automated attacks or scans in general) by limiting the number of HTTP requests per unit of time coming from the same IP address. If the rate of requests exceeds a certain threshold, the web server will respond with the status code 429 ("Too Many Requests") for a period of time between seconds and minutes. Of course, this IP address block can be easily bypassed by attacking the elevator management site through a proxy server, and automatically change the proxy when its IP address is blocked. As a proof of concept, we developed a Python script which submits several passwords from a specified wordlist, and cyclically rotates the proxy from a configurable list of active proxies when a 429 status code is received. The script can also run several threads in parallel to improve the throughput of the brute-force attack.

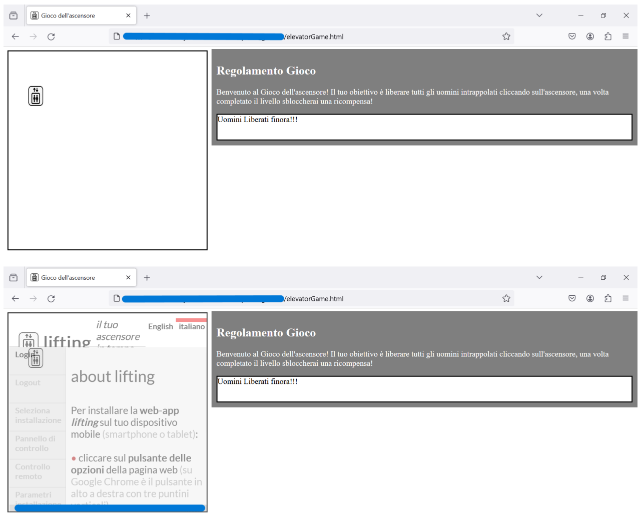
Figure 2 shows the performance of the described attack in terms of number of passwords tried per minute, compared to the classic Hydra tool [19]. Hydra cannot use proxy rotation, so we had to downtune it to make a login attempt every 10 seconds, in such a way to avoid the IP block. We configure our brute-force script to use 1 thread with 10 proxies, or 10 parallel threads with 10 proxies. With the latter configuration we reach 147 login attempts per minute, which corresponds to more than 211K passwords per day, and we completely avoid the IP block. If the passwords are not strong enough, this brute-force attack could guess one of them in a reasonable amount of time. To avoid this, the web site should implement a functionality that blocks a username when it attempts too many failed logins per unit of time. A username-based block is more difficult to bypass than an IP-based one. Indeed, in order to change the attacked username, the attacker should have a list of valid usernames. Moreover, even if the attacker has such a list, he must restart the password brute forcing from scratch every time he changes the attacked username.

6 Cookie Security and Clickjacking

After successful authentication, the elevator management site releases a random session token to the user through the customary `PHPSESSID` cookie of the PHP language, to re-identify him through the various requests. From our tests, it turned out that no cookie (neither the `PHPSESSID` cookie as well) has the `Secure` nor the `HttpOnly` flags set, and no cookie has the `SameSite` attribute set. The `Secure` flag tells the browser not to transmit the cookie in requests performed over unencrypted connections. Such a flag is important to avoid that the browser discloses the session token to a man-in-the-middle attacker if the user requests a page of the unencrypted version of the site (i.e., the `http://` one). Note that it does not matter whether such an unencrypted version exists or not, because the session token is compromised just as soon as the user performs the request. The site developers should set the `Secure` flag, at least for the `PHPSESSID` cookie, as a free defense in depth against man-in-the-middle attacks.

The absence of the `HttpOnly` flag means that the cookie is accessible to client-side scripts (e.g., JavaScript). Such a flag is a defense in depth against cross-site scripting (XSS) attacks, in which the attacker injects a client-side script leveraging a bug in the back-end logic. The injected script typically steals the `PHPSESSID` cookie of a legitimate user by submitting it to a site controlled by the attacker, which can then hijack the ongoing session of the user. After extensive tests, no evident XSS bug has been found on the site. Indeed, due to the nature of the site, user input is rarely used to construct return pages. Nevertheless, the elevator management site should set the `PHPSESSID` cookie as `HttpOnly` as a free defense in depth against XSS.

Finally, the `SameSite` attribute tells the browser when to transmit the cookie in case of cross-site requests. With the `None` value, the cookie will be always sent within all requests. With the `Strict` value, the cookie will be sent only within in-site requests, and not within cross-site ones. With the `Lax` value, the cookie will be sent in cross-site requests only if the request method is a safe one (e.g., GET, but not POST) or if the request is triggered by a top-level navigation event (e.g., clicking on a link, but not loading an iframe). Since 2020, the most widespread browsers decided to use `Lax` rather than `None` as the default value for the `SameSite` attribute, thus forbidding by default the transmission of cookies within dangerous cross-site requests. This helps counteracting cross-site request forgery (CSRF) attacks in new browsers. However, an outdated browser will automatically attach the cookie to cross-site requests, for example when a legitimate user follow a malicious link or submits a malicious form, or when he navigates a malicious site that contains the elevator management site as an iframe. Even here, the elevator management site should explicitly set the `PHPSESSID` cookie as `SameSite=Strict` as a free defense in depth against CSRF. We noted also that an explicit protection against CSRF is implemented by the site, by means of an anti-CSRF token. The implementation



Regolamento Gioco

Benvenuto al Gioco dell'ascensore! Il tuo obiettivo è liberare tutti gli uomini intrappolati cliccando sull'ascensore, una volta completato il livello sbloccherai una ricompensa!

Uomini Liberati finora!!!

about lifting

Per installare la **web-app** ***lifting*** sul tuo dispositivo mobile (smartphone o tablet):

● cliccare sul **pulsante delle opzioni** della pagina web (su Google Chrome è il pulsante in alto a destra con tre puntini

Figure 3: Malicious clickjacker site, opaque version (above) for attack deployment, and semi-transparent version (below) for attack debug

of such an anti-CSRF token shows some weaknesses, that will be analyzed in Section 7.

From further tests, it emerged that the elevator management site did not use any Content Security Policy (CSP) directive, particularly those intended to mitigate XSS and clickjacking. To test the feasibility of the latter attack, we developed a malicious web site that frames the elevator management site into an `<iframe>` HTML element, and it induces a legitimate user to perform unwitting actions on it. Our clickjacker web site was built by extending the code automatically outputted by the Clickbandit tool [2]. The malicious decoy site is apparently a simple video game. The user is asked to click on an icon of an elevator that moves within the game area, to "free" some men trapped in an elevator. Clicking on the elevator icon causes a click on the underlying (and hidden) frame containing the original elevator management site. After each click, the elevator icon moves in a different place inside the game area, in such a way to steer the user's clicks on different links and buttons of the original site. In practice, the user thinks he is playing a game about elevators, while he is sending commands to the real elevators, possibly causing damage and harm. Figure 3 shows two screenshots of the malicious game. In the second screenshot the game area was purposefully made semi-transparent to show the underlying original site.

To perform the attack we had to induce the user to authenticate with his password to the framed site. However, this was quite easy to do by leveraging the password autocompletion mechanism of the browser, which works even if the web site is framed into another one. The hardest technical issue to address was detecting the user's clicks, in order to move the elevator icon and make the game progress. Adding a listener directly on the iframe to detect clicks is not possible due to same-origin policy restrictions, which prevent a web document's code from directly accessing the content of an iframe loaded from a different site. We tried to evade the restriction by taking a screenshot of the web page with the html2canvas JavaScript library. However, the same-origin policy again made the trick ineffective. Finally, we solved the problem by intercepting the event of the mouse pointer being over the game area, after which we forcefully move the focus on an invisible "clickjack focus" input. Then, we intercept the successive event of the focus moving away from it. This sequence of events can be easily intercepted, and it indicates that a click occurred on the framed original site[1].

To avoid the clickjacking attack, the elevator management site should attach a `frame-ancestors` CSP directive with a value of `none` to each returned page. In this way, the client's browser is not allowed to embed any page of the site within an iframe or similar embedding HTML elements.

7 Anti-CSRF Token Issues

In addition to the `PHPSESSID` cookie, the elevator management site uses an URL parameter named "`x`" to control the page sequence and to re-identify the user through different requests. The parameter is a base64-encoded JSON object, which contains three fields named respectively "`aid`", "`lang`", and "`auth`". The first two fields identify respectively the page requested by the user and the language of such a page. The `x.auth` field is an anti-CSRF token [3], which is randomly chosen by the server after the authentication, saved as a session variable at the server side, and checked again at each request by the server.

We noted that the anti-CSRF token is submitted and checked by the server also in case of innocuous GET requests, which suggests us that developers could be not fully aware of the real purpose of the token. More crucially, using an URL parameter as anti-CSRF token is a bad practice, because URLs are not designed to keep secrets in general. Indeed, the requested URLs can be revealed by the `Referer` header in cross-site requests. They are also logged by the server, by possible proxies and load balancers, and by the browser itself in its navigation history. The elevator management site should transmit the anti-CSRF token in a more secure way, for example as a parameter in the request body or in the request headers.

We found more weaknesses in the way the anti-CSRF is managed at the server side. Fortunately, none of these weaknesses is immediately exploitable, but they could become so in future versions of the site. For example, the token is created by the `mcrypt_create_iv()` PHP function, which has been deprecated since PHP 7.1 and removed since PHP 7.2 due to security concerns and lack of maintenance. The site should instead use the `openssl_random_pseudo_bytes()` function, which is based on the OpenSSL library. More interestingly, the anti-CSRF token is checked as shown in Figure 4.
We first note that the `strcmp()` function does not run in constant time, but rather its running time depends on the position of the first differing character. This could lead to possible side-channel attacks to discover the anti-CSRF token [6]. However, these attacks are rarely exploitable in a real web application, because the network and processing delays are so variable that they make the tiny timing differences in the token comparison extremely difficult to measure.

[1]This trick does not work if the victim user uses Firefox, because a click on the iframe does not cause the "clickjack focus" input to lose focus.

```
if (strlen($submitted_token) > 0) {
  if (strcmp($submitted_token, $real_token) == 0) {
    \\ perform the action and return the page
  }
}
```

Figure 4: Code that checks the anti-CSRF token

However, other weaknesses are present in the code of Figure 4. Since the token is passed as a JSON object's field, it can have a type different from string, leading to possible PHP type juggling vulnerabilities. In particular, submitting an array (`array()`) as token leads `strcmp()` to produce `null` independently from the real token value, which in turn produces `true` when loosely compared (note the == operator) to `0`. This could allow an attacker to bypass the CSRF defense by simply submitting `array()` as anti-CSRF token in the URL. This behavior is prevented in PHP 8.0, because `strcmp()` raises an exception in case one of the two arguments is not a string. However, the elevator management site runs over PHP 5.6, therefore `strcmp()` only raises a warning in this case, and then it continues executing the code.

For a fortuitous coincidence, the `strcmp()` invocation is preceded by a `strlen()` invocation, which was put there to check whether the user jumped on that page before authentication. In case the argument is not a string, the `strlen()` function also produces `null`. For the type juggling rules, `null` is converted into `0` before the `>0` comparison, which in turn returns `false`. In other words, by checking that the length of the submitted anti-CSRF token is greater than zero, the site prevents the `strcmp()` bug from being triggered, and thus it luckily defends against type juggling attack. To defend more securely, the site should check the existence and the type of the anti-CSRF token before using any string function, respectively by means of the `isset()` and `is_string()` functions. Then, it should check the `strcmp()`'s result with a strict comparison (===). Using a more recent version of PHP would also serve as a defense in depth.

8 Conclusions

In this paper we carried out a white-box penetration test of the web site deputed to the monitoring and management of a CPS composed of a series of elevators deployed in non-industrial buildings. The security of the management web site is critical, because it is the CPS component most exposed to possible attacks due to its public and remote availability. From our experience we can conclude that, for what it concerns the web part, the CPS developers have a good awareness of the most common cybersecurity threats, and they a aware of common defense techniques like anti-CSRF tokens [3]. Still, although we did not find any major and easily exploitable vulnerability, some defenses against advanced client-side attacks like clickjacking [2] are missing. Developers adopted some defense techniques without following the best practices (e.g., they put secrets in URLs), and without being aware of some tricky server-side language functionalities (e.g., PHP type juggling [15]). They also do not employ some free defense-in-depth mechanisms, which could give more security without any impact on performance or complexity. This negligence could lead to vulnerabilities in future versions of the site, or in case some unfortunate conditions are met.

Acknowledgments

Work partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union – NextGenerationEU.

References

[1] PortSwigger Web Security Academy. 2023. *SQL Injection.* https://portswigger.net/web-security/sql-injection

[2] PortSwigger Web Security Academy. 2024. *Clickjacking (UI redressing).* https://portswigger.net/web-security/clickjacking

[3] PortSwigger Web Security Academy. 2024. *How to prevent CSRF vulnerabilities.* https://portswigger.net/web-security/csrf/preventing

[4] Lucas Apa. 2017. *Exploiting Industrial Collaborative Robots.* https://ioactive.com/exploiting-industrial-collaborative-robots/

[5] Deval Bhamare, Maede Zolanvari, Aiman Erbad, Raj Jain, Khaled Khan, and Nader Meskin. 2020. Cybersecurity for industrial control systems: A survey. *Computers & Security* 89 (2020), 101677. https://doi.org/10.1016/j.cose.2019.101677

[6] David Brumley and Dan Boneh. 2005. Remote timing attacks are practical. *Computer Networks* 48, 5 (2005), 701–716. https://doi.org/10.1016/j.comnet.2005.01.010 Web Security.

[7] Martin Brunner, Hans Hofinger, Christoph Krauß, Christopher Roblee, Peter Schoo, and Sascha Todt. 2010. Infiltrating critical infrastructures with next-generation attacks. *Fraunhofer Institute for Secure Information Technology (SIT), Munich* (2010).

[8] OWASP Foundation. 2024. *Cross Site Request Forgery (CSRF).* https://owasp.org/www-community/attacks/csrf

[9] OWASP Foundation. 2024. *SQL Injection Prevention Cheat Sheet.* https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[10] The SecOps Group. 2022. *PHP Type Juggling Simplified.* https://secops.group/php-type-juggling-simplified/

[11] Song Han, Miao Xie, Hsiao-Hwa Chen, and Yun Ling. 2014. Intrusion Detection in Cyber-Physical Systems: Techniques and Challenges. *IEEE Systems Journal* 8, 4 (2014), 1052–1062. https://doi.org/10.1109/JSYST.2013.2257594

[12] Christopher Harper. 2024. *Biggest password database posted in history spills 10 billion passwords — RockYou2024 is a massive compilation of known passwords.* https://www.tomshardware.com/tech-industry/cyber-security/biggest-password-leak-in-history-spills-10-billion-passwords

[13] Haider Adnan Khan, Nader Sehatbakhsh, Luong N. Nguyen, Robert L. Callan, Arie Yeredor, Milos Prvulovic, and Alenka Zajić. 2021. IDEA: Intrusion Detection through Electromagnetic-Signal Analysis for Critical Embedded and Cyber-Physical Systems. *IEEE Transactions on Dependable and Secure Computing* 18, 3 (2021), 1150–1163. https://doi.org/10.1109/TDSC.2019.2932736

[14] Hongwei Li and Danai Chasaki. 2021. Ensemble Machine Learning for Intrusion Detection in Cyber-Physical Systems. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).* 1–2. https://doi.org/10.1109/INFOCOMWKSHPS51825.2021.9484479

[15] PHP Manual. 2024. *Type Juggling.* https://www.php.net/manual/en/language.types.type-juggling.php

[16] Marcello Pogliani, Davide Quarta, Mario Polino, Martino Vittone, Federico Maggi, and Stefano Zanero. 2019. Security of controlled manufacturing systems in the connected factory: the case of industrial robots. *Journal of Computer Virology and Hacking Techniques* 15 (09 2019). https://doi.org/10.1007/s11416-019-00329-8

[17] Niels Provos and David Mazières. 1999. A future-adaptive password scheme. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference* (Monterey, California) *(ATEC '99).* USENIX Association, USA, 32.

[18] Davide Quarta, Marcello Pogliani, Mario Polino, Federico Maggi, Andrea Maria Zanchettin, and Stefano Zanero. 2017. An Experimental Security Analysis of an Industrial Robot Controller. In *2017 IEEE Symposium on Security and Privacy (SP).* 268–286. https://doi.org/10.1109/SP.2017.20

[19] van Hauser. 2023. *Hydra.* https://github.com/vanhauser-thc/thc-hydra

[20] Wenzhuo Yang, Zhaowei Chu, Jiani Fan, Ziyao Liu, and Kwok-Yan Lam. 2023. A Practical Intrusion Detection System Trained on Ambiguously Labeled Data for Enhancing IIoT Security. In *Proceedings of the 9th ACM Cyber-Physical System Security Workshop* (Melbourne, VIC, Australia) *(CPSS '23).* Association for Computing Machinery, New York, NY, USA, 14–23. https://doi.org/10.1145/3592538.3594270