OPIS PRZEDMIOTU ZAMÓWIENIA

I. Przedmiotem zamówienia jest usługa przeprowadzenia manualnych testów bezpieczeństwa/penetracyjnych dla usługi Portal Informacyjny Sądów Powszechnych dostępnej dla użytkowników poprzez sieć Internet oraz Panelu Administracyjnego działającego w sieci WAN. Testy mają objąć: **aplikacje Web (aplikacja Portal Informacyjny [Internet] oraz aplikacja Panel Administracyjny [sieć WAN]), aplikacje mobilne (Android, iOS) i ich interfejsy API.**

II. Zakres minimalny testów:

1. dla aplikacji Web / interfejsu API systemu:

Testy aplikacji WWW według standardu OWASP ASVS 4.0 Level 1 – Greybox (bez dostępu do systemu operacyjnego i kodu źródłowego):

1) Verify that user set passwords are at least 12 characters in length. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
2) Verify that there are no periodic credential rotation or password history requirements.
3) Verify that "paste" functionality, browser password helpers, and external password managers are permitted.
4) Verify that the user can choose to either temporarily view the entire masked password, or temporarily view the last typed character of the password on platforms that do not have this as native functionality.
5) Verify that passwords 64 characters or longer are permitted. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
6) Verify that passwords can contain spaces and truncation is not performed. Consecutive multiple spaces MAY optionally be coalesced. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
7) Verify that Unicode characters are permitted in passwords. A single Unicode code point is considered a character, so 12 emoji or 64 kanji characters should be valid and permitted.
8) Verify users can change their password.
9) Verify that password change functionality requires the user's current and new password.
10) Verify that passwords submitted during account registration, login, and password change are checked against a set of breached passwords either locally (such as the top 1,000 or 10,000 most common passwords which match the system's password policy) or using an external API. If using an API a zero knowledge proof or other mechanism should be used to ensure that the plain text password is not sent or used in verifying the breach status of the password. If the password is breached, the application must require the user to set a new non-breached password. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
11) Verify that a password strength meter is provided to help users set a stronger password.
12) Verify that there are no password composition rules limiting the type of characters permitted. There should be no requirement for upper or lower case or numbers or special characters. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
13) Verify that anti-automation controls are effective at mitigating breached credential testing, brute force, and account lockout attacks. Such controls include blocking the most common breached passwords, soft lockouts, rate limiting, CAPTCHA, ever increasing delays between attempts, IP address restrictions, or risk-based restrictions such as location, first login on a device, recent attempts to unlock the account, or similar. Verify that no more than 100 failed attempts per hour is possible on a single account.
14) Verify that the use of weak authenticators (such as SMS and email) is limited to secondary verification and transaction approval and not as a replacement for more secure authentication methods. Verify that stronger methods are offered before weak methods, users are aware of the risks, or that proper measures are in place to limit the risks of account compromise.
15) Verify that secure notifications are sent to users after updates to authentication details, such as credential resets, email or address changes, logging in from unknown or risky locations. The use of push notifications - rather than SMS or email - is preferred, but in the absence of push notifications, SMS or email is acceptable as long as no sensitive information is disclosed in the notification.

16) Verify system generated initial passwords or activation codes SHOULD be securely randomly generated, SHOULD be at least 6 characters long, and MAY contain letters and numbers, and expire after a short period of time. These initial secrets must not be permitted to become the long term password.
17) Verify that a system generated initial activation or recovery secret is not sent in clear text to the user. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
18) Verify password hints or knowledge-based authentication (so-called "secret questions") are not present.
19) Verify password credential recovery does not reveal the current password in any way. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
20) Verify shared or default accounts are not present (e.g. "root", "admin", or "sa").
21) Verify that if an authentication factor is changed or replaced, that the user is notified of this event.
22) Verify forgotten password, and other recovery paths use a secure recovery mechanism, such as TOTP or other soft token, mobile push, or another offline recovery mechanism. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
23) Verify that clear text out of band (NIST "restricted") authenticators, such as SMS or PSTN, are not offered by default, and stronger alternatives such as push notifications are offered first.
24) Verify that the out of band verifier expires out of band authentication requests, codes, or tokens after 10 minutes.
25) Verify that the out of band verifier authentication requests, codes, or tokens are only usable once, and only for the original authentication request.
26) Verify that the out of band authenticator and verifier communicates over a secure independent channel.
27) Verify that time-based OTPs have a defined lifetime before expiring.
28) Verify the application never reveals session tokens in URL parameters or error messages.
29) Verify the application generates a new session token on user authentication. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
30) Verify that session tokens possess at least 64 bits of entropy. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
31) Verify the application only stores session tokens in the browser using secure methods such as appropriately secured cookies (see section 3.4) or HTML 5 session storage.
32) Verify that logout and expiration invalidate the session token, such that the back button or a downstream relying party does not resume an authenticated session, including across relying parties. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
33) Verify that cookie-based session tokens have the 'Secure' attribute set. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
34) Verify that cookie-based session tokens have the 'HttpOnly' attribute set. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
35) Verify that cookie-based session tokens utilize the 'SameSite' attribute to limit exposure to cross-site request forgery attacks. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
36) Verify that cookie-based session tokens use "__Host-" prefix (see references) to provide session cookie confidentiality.
37) Verify that if the application is published under a domain name with other applications that set or use session cookies that might override or disclose the session cookies, set the path attribute in cookie-based session tokens using the most precise path possible. ([C6](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
38) Verify the application ensures a valid login session or requires re-authentication or secondary verification before allowing any sensitive transactions or account modifications.
39) Verify that the application enforces access control rules on a trusted service layer, especially if client-side access control is present and could be bypassed.
40) Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.
41) Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection

against spoofing and elevation of privilege. ([C7](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

42) Verify that the principle of deny by default exists whereby new users/roles start with minimal or no permissions and users/roles do not receive access to new features until access is explicitly assigned. ([C7](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

43) Verify that access controls fail securely including when an exception occurs. ([C10](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

44) Verify that sensitive data and APIs are protected against direct object attacks targeting creation, reading, updating and deletion of records, such as creating or updating someone else's record, viewing everyone's records, or deleting all records.

45) Verify that the application or framework enforces a strong anti-CSRF mechanism to protect authenticated functionality, and effective anti-automation or anti-CSRF protects unauthenticated functionality.

46) Verify administrative interfaces use appropriate multi-factor authentication to prevent unauthorized use.

47) Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.

48) Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).

49) Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar. ([C5](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

50) Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting). ([C5](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

51) Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as checking that suburb and zip/postcode match). ([C5](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

52) Verify that URL redirects and forwards only allow whitelisted destinations, or show a warning when redirecting to potentially untrusted content.

53) Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. ([C5](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

54) Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.

55) Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.

56) Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being included must be sanitized or sandboxed before being executed.

57) Verify that the application protects against template injection attacks by ensuring that any user input being included is sanitized or sandboxed.

58) Verify that the application protects against SSRF attacks, by validating or sanitizing untrusted data or HTTP file metadata, such as filenames and URL input fields, use whitelisting of protocols, domains, paths and ports.

59) Verify that the application sanitizes, disables, or sandboxes user-supplied SVG scriptable content, especially as they relate to XSS resulting from inline scripts, and foreignObject.

60) Verify that the application sanitizes, disables, or sandboxes user-supplied scriptable or expression template language content, such as Markdown, CSS or XSL stylesheets, BBCode, or similar.

61) Verify that output encoding is relevant for the interpreter and context required. For example, use encoders specifically for HTML values, HTML attributes, JavaScript, URL Parameters, HTTP headers, SMTP, and others as the context requires, especially from untrusted inputs (e.g. names with Unicode or apostrophes, such as ねこ or O'Hara). ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

62) Verify that the application protects against XPath injection or XML injection attacks. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))

63) Verify that output encoding preserves the user's chosen character set and locale, such that any Unicode character point is valid and safely handled. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
64) Verify that context-aware, preferably automated - or at worst, manual - output escaping protects against reflected, stored, and DOM based XSS. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
65) Verify that data selection or database queries (e.g. SQL, HQL, ORM, NoSQL) use parameterized queries, ORMs, entity frameworks, or are otherwise protected from database injection attacks. ([C3](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
66) Verify that where parameterized or safer mechanisms are not present, context-specific output encoding is used to protect against injection attacks, such as the use of SQL escaping to protect against SQL injection. ([C3, C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
67) Verify that the application projects against JavaScript or JSON injection attacks, including for eval attacks, remote JavaScript includes, CSP bypasses, DOM XSS, and JavaScript expression evaluation. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
68) Verify that the application protects against LDAP Injection vulnerabilities, or that specific security controls to prevent LDAP Injection have been implemented. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
69) Verify that the application protects against OS command injection and that operating system calls use parameterized OS queries or use contextual command line output encoding. ([C4](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
70) Verify that the application protects against Local File Inclusion (LFI) or Remote File Inclusion (RFI) attacks.
71) Verify that serialized objects use integrity checks or are encrypted to prevent hostile object creation or data tampering. ([C5](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
72) Verify that the application correctly restricts XML parsers to only use the most restrictive configuration possible and to ensure that unsafe features such as resolving external entities are disabled to prevent XXE.
73) Verify that deserialization of untrusted data is avoided or is protected in both custom code and third-party libraries (such as JSON, XML and YAML parsers).
74) Verify that when parsing JSON in browsers or JavaScript-based backends, JSON.parse is used to parse the JSON document. Do not use eval() to parse JSON.
75) Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks.
76) Verify that the application does not log credentials or payment details. Session tokens should only be stored in logs in an irreversible, hashed form. ([C9, C10](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
77) Verify that the application does not log other sensitive data as defined under local privacy laws or relevant security policy. ([C9](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
78) Verify that a generic message is shown when an unexpected or security sensitive error occurs, potentially with a unique ID which support personnel can use to investigate. ([C10](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
79) Verify the application sets sufficient anti-caching headers so that sensitive data is not cached in modern browsers.
80) Verify that data stored in client side storage (such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies) does not contain sensitive data or PII.
81) Verify that authenticated data is cleared from client storage, such as the browser DOM, after the client or session is terminated.
82) Verify that sensitive data is sent to the server in the HTTP message body or headers, and that query string parameters from any HTTP verb do not contain sensitive data.
83) Verify that users have a method to remove or export their data on demand.
84) Verify that users are provided clear language regarding collection and use of supplied personal information and that users have provided opt-in consent for the use of that data before it is used in any way.

85) Verify that all sensitive data created and processed by the application has been identified, and ensure that a policy is in place on how to deal with sensitive data. ([C8](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
86) Verify that secured TLS is used for all client connectivity, and does not fall back to insecure or unencrypted protocols. ([C8](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
87) Verify using online or up to date TLS testing tools that only strong algorithms, ciphers, and protocols are enabled, with the strongest algorithms and ciphers set as preferred.
88) Verify that old versions of SSL and TLS protocols, algorithms, ciphers, and configuration are disabled, such as SSLv2, SSLv3, or TLS 1.0 and TLS 1.1. The latest version of TLS should be the preferred cipher suite.
89) Verify that if the application has a client or server auto-update feature, updates should be obtained over secure channels and digitally signed. The update code must validate the digital signature of the update before installing or executing the update.
90) Verify that the application employs integrity protections, such as code signing or sub-resource integrity. The application must not load or execute code from untrusted sources, such as loading includes, modules, plugins, code, or libraries from untrusted sources or the Internet.
91) Verify that the application has protection from sub-domain takeovers if the application relies upon DNS entries or DNS sub-domains, such as expired domain names, out of date DNS pointers or CNAMEs, expired projects at public source code repos, or transient cloud APIs, serverless functions, or storage buckets (autogen-bucket-id.cloud.example.com) or similar. Protections can include ensuring that DNS names used by applications are regularly checked for expiry or change.
92) Verify the application will only process business logic flows for the same user in sequential step order and without skipping steps.
93) Verify the application will only process business logic flows with all steps being processed in realistic human time, i.e. transactions are not submitted too quickly.
94) Verify the application has appropriate limits for specific business actions or transactions which are correctly enforced on a per user basis.
95) Verify the application has sufficient anti-automation controls to detect and protect against data exfiltration, excessive business logic requests, excessive file uploads or denial of service attacks.
96) Verify the application has business logic limits or validation to protect against likely business risks or threats, identified using threat modelling or similar methodologies.
97) Verify that the application will not accept large files that could fill up storage or cause a denial of service attack.
98) Verify that user-submitted filename metadata is not used directly with system or framework file and URL API to protect against path traversal.
99) Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure, creation, updating or removal of local files (LFI).
100) Verify that user-submitted filename metadata is validated or ignored to prevent the disclosure or execution of remote files (RFI), which may also lead to SSRF.
101) Verify that the application protects against reflective file download (RFD) by validating or ignoring user-submitted filenames in a JSON, JSONP, or URL parameter, the response Content-Type header should be set to text/plain, and the Content-Disposition header should have a fixed filename.
102) Verify that untrusted file metadata is not used directly with system API or libraries, to protect against OS command injection.
103) Verify that files obtained from untrusted sources are stored outside the web root, with limited permissions, preferably with strong validation.
104) Verify that files obtained from untrusted sources are scanned by antivirus scanners to prevent upload of known malicious content.
105) Verify that the web tier is configured to serve only files with specific file extensions to prevent unintentional information and source code leakage. For example, backup files (e.g. .bak), temporary working files (e.g. .swp), compressed files (.zip, .tar.gz, etc) and other extensions commonly used by editors should be blocked unless required.
106) Verify that direct requests to uploaded files will never be executed as HTML/JavaScript content.
107) Verify that the web or application server is configured with a whitelist of resources or systems to which the server can send requests or load data/files from.

108) Verify that all application components use the same encodings and parsers to avoid parsing attacks that exploit different URI or file parsing behavior that could be used in SSRF and RFI attacks.
109) Verify that access to administration and management functions is limited to authorized administrators.
110) Verify API URLs do not expose sensitive information, such as the API key, session tokens etc.
111) Verify that enabled RESTful HTTP methods are a valid choice for the user or action, such as preventing normal users using DELETE or PUT on protected API or resources.
112) Verify that JSON schema validation is in place and verified before accepting input.
113) Verify that RESTful web services that utilize cookies are protected from Cross-Site Request Forgery via the use of at least one or more of the following: triple or double submit cookie pattern (see [references](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)), CSRF nonces, or ORIGIN request header checks.
114) Verify that XSD schema validation takes place to ensure a properly formed XML document, followed by validation of each input field before any processing of that data takes place.
115) Verify that all components are up to date, preferably using a dependency checker during build or compile time. ([C2](https://www.owasp.org/index.php/OWASP_Proactive_Controls#tab=Formal_Numbering))
116) Verify that all unneeded features, documentation, samples, configurations are removed, such as sample applications, platform documentation, and default or example users.
117) Verify that if application assets, such as JavaScript libraries, CSS stylesheets or web fonts, are hosted externally on a content delivery network (CDN) or external provider, Subresource Integrity (SRI) is used to validate the integrity of the asset.
118) Verify that web or application server and framework error messages are configured to deliver user actionable, customized responses to eliminate any unintended security disclosures.
119) Verify that web or application server and application framework debug modes are disabled in production to eliminate debug features, developer consoles, and unintended security disclosures.
120) Verify that the HTTP headers or any part of the HTTP response do not expose detailed version information of system components.
121) Verify that every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8, ISO 8859-1).
122) Verify that all API responses contain Content-Disposition: attachment; filename="api.json" (or other appropriate filename for the content type).
123) Verify that a content security policy (CSPv2) is in place that helps mitigate impact for XSS attacks like HTML, DOM, JSON, and JavaScript injection vulnerabilities.
124) Verify that all responses contain X-Content-Type-Options: nosniff.
125) Verify that HTTP Strict Transport Security headers are included on all responses and for all subdomains, such as Strict-Transport-Security: max-age=15724800; includeSubdomains.
126) Verify that a suitable "Referrer-Policy" header is included, such as "no-referrer" or "same-origin".
127) Verify that a suitable X-Frame-Options or Content-Security-Policy: frame-ancestors header is in use for sites where content should not be embedded in a third-party site.
128) Verify that the application server only accepts the HTTP methods in use by the application or API, including pre-flight OPTIONS.
129) Verify that the supplied Origin header is not used for authentication or access control decisions, as the Origin header can easily be changed by an attacker.
130) Verify that the cross-domain resource sharing (CORS) Access-Control-Allow-Origin header uses a strict white-list of trusted domains to match against and does not support the "null" origin.

2. Dla aplikacji mobilnej:

1) Zgodnie z OWASP Mobile Application Security Checklist version 1.1.2 dla Androida i iOS. Zakres testów wymagany przez Zamawiającego w załączeniu (Mobile_App_Security_Checklist_1.1.2.xlsx), przy czym testy nie będą obejmowały zakresu Anti-Re - Android i Anti-RE - iOS (zakres Anti-Re - Android i Anti-RE podlega wyłączeniu).
2) Aplikacja mobilna dostępna jest:
   a) na Androida: w sklepie Google Play, nazwa Portal Informacyjny
   b) na iOS: w sklepie App Store, nazwa Portal Informacyjny

III. **W celu przeprowadzenia testów:**

a) Wykonawca otrzyma możliwość dostępu z komputera lub telefonu Wykonawcy do testowanego środowiska, co potwierdzi Zamawiającemu na zasadach określonych w umowie, której wzór stanowi załącznik do zapytania.

b) Zamawiający utworzy w testowanych aplikacjach konto/konta z różnymi uprawnieniami (odpowiadającymi rolom w tych systemach) oraz udostępnieni Wykonawcy dane uwierzytelniające do tych kont.

c) Zamawiający udostępnieni dane do systemowych kont administracyjnych badanego środowiska.

d) Zamawiający umożliwi nawiązanie z komputera lub telefonu Wykonawcy połączeń sieciowych z badanym środowiskiem (np. SSH) na zasadach ustalonych miedzy stronami.

e) Zamawiający umożliwi uruchomienie na badanych komponentach systemu skryptów Wykonawcy z uprawnieniami administracyjnymi.

f) Zamawiający zapewni możliwość przywrócenia testowanego środowiska w przypadku jego uszkodzenia podczas testów, pod warunkiem, że testy zostaną wykonane w ustalonym wcześniej przez Strony oknie serwisowym (oknie ustalonym dla bezpiecznego wykonania testów), a Wykonawca będzie dochowywał należytej staranności, aby do uszkodzenia testowanego środowiska nie doszło.

IV. **Zamawiający wymaga**, aby Wykonawca zrealizował testy objęte niniejszym zamówieniem w sposób manualny, przy czym testy realizowane w sposób manualny rozumiane są jako testy przeprowadzone ręcznie przez testującego z odpowiednim doświadczeniem i wiedzą w przeprowadzaniu takich testów.

V. **Wynikiem przeprowadzenia testów** będzie raport opracowany w języku polskim (ewentualnie w części ustalonej przez Strony na spotkaniu technicznym, w języku angielskim) i przedstawiony Zamawiającemu przez Wykonawcę w terminie i na zasadach wskazanych w Umowie. Raport musi zawierać co najmniej poniższe informacje:

g) o wykonanym teście wraz z opisem (rodzaj, metodyka, itp.);

h) wyniki testu w odpowiedniej, przyjętej dla tego rodzaju testów, skali wraz z komentarzem i zaleceniami, tj. wyniki testu muszą zawierać zidentyfikowane podatności wraz z ich opisem, skalą zagrożenia (krytyczności) dla bezpieczeństwa testowanych usług oraz rekomendacjami w zakresie usunięcia lub zminimalizowania podatności.

VI. **Termin realizacji zamówienia:** 30 dni. Termin ten liczony jest od dnia otrzymania przez Wykonawcę ostatniego dostępu do testowanego środowiska.

Informacje dodatkowe

1. Zamawiający wymaga, aby na potwierdzenie doświadczenia Wykonawcy w prowadzeniu testów tego rodzaju, przedstawił on do oferty referencje co najmniej dwóch testerów posiadających odpowiednią wiedzę i doświadczenie w przeprowadzaniu manualnych testów penetracyjnych aplikacji webowych lub aplikacji mobilnych. Tester musi posiadać co najmniej 3-letnie doświadczenie w przeprowadzaniu manualnych testów penetracyjnych oraz posiadać wiedzę potwierdzoną co najmniej 2 certyfikatami:
   Tester wiodący min.:
   a. Certyfikat Offensive Security Certified Expert (OSCE)
   b. Certyfikat CompTIA Security+
   Tester wspierający min.:
   c. Certyfikat Certified Ethical Hacker
   d. Certyfikat CompTIA Security+
   lub równoważnymi, przy czym za równoważne Zamawiający uzna certyfikaty wystawione przez niezależny od Wykonawcy podmiot, specjalizujący się w certyfikacji wiedzy z obszaru co najmniej bezpieczeństwa informatycznego potwierdzający posiadanie wiedzy na nie niższym poziomie, z co najmniej tego samego zakresu

2. Zamawiający wymaga, aby Strony przed przystąpieniem do realizacji testów wzięły udział w tzw. spotkaniu technicznym w celu szczegółowego określenia metodyki pracy i zasad współpracy w zakresie przeprowadzenia przedmiotowych testów. Zamawiający wymaga, aby ze strony Wykonawcy w spotkaniu wzięły udział osoby wskazane w liście testerów skierowanych do realizacji testów (Wykaz osób stanowiący Załącznik do Umowy). Spotkanie może odbyć się w siedzibie Zamawiającego (w granicach administracyjnych Wrocławia) lub poprzez wideokonferencję w terminie ustalonym przez Strony.