15 punktów bezpieczeństwa Twojej aplikacji zbudowanej z AI

Praktyczna checklista bezpieczeństwa dla osób, które tworzą własne aplikacje i sklepy internetowe metodą „vibe-coding", czyli programowania z pomocą sztucznej inteligencji (np. Cursor, Lovable, Replit, ChatGPT czy Claude).

Checklista bezpieczeństwa aplikacji

Poziom podstawowy

- [ ] 1. Klucze i hasła „zaszyte" w kodzie
- [ ] 2. Panel administracyjny pod przewidywalnym adresem
- [ ] 3. Formularze bez podstawowej walidacji danych
- [ ] 4. Brak certyfikatu SSL (HTTPS)
- [ ] 5. Nieaktualizowane biblioteki i zależności

Poziom średniozaawansowany

- [ ] 6. Czy widzisz dane innych użytkowników, zmieniając numer w adresie (IDOR)
- [ ] 7. Zarządzanie sesją i wylogowaniem
- [ ] 8. Uprawnienia dostępu do bazy danych
- [ ] 9. Konfiguracja CORS: kto może „rozmawiać" z Twoim backendem
- [ ] 10. Ochrona przed automatycznymi atakami (rate limiting, captcha)

Poziom zaawansowany

- [ ] 11. Testowanie podatności typu SQL Injection i XSS
- [ ] 12. Przegląd kodu pod kątem bezpieczeństwa (SAST)
- [ ] 13. Zabezpieczenia infrastruktury i hostingu
- [ ] 14. Bezpieczeństwo łańcucha dostaw (biblioteki i integracje)
- [ ] 15. Plan reagowania na incydent bezpieczeństwa

Dziesięć z piętnastu punktów sprawdzisz samodzielnie, bez wiedzy technicznej.

Dlaczego „działa" nie oznacza, że „jest bezpieczne"

Jeśli zbudowałeś swoją aplikację albo sklep internetowy, "rozmawiając" z AI - opisując mu, co ma zrobić, i patrząc, jak samo pisze kod - jesteś w bardzo dobrym towarzystwie. To podejście, nazywane potocznie vibe-codingiem (programowaniem "na wyczucie", gdzie AI pisze kod na podstawie Twoich opisów, a Ty oceniasz efekt, nie zawsze rozumiejąc każdą linijkę), pozwoliło tysiącom właścicieli małych firm postawić własny produkt cyfrowy bez zatrudniania programisty. To naprawdę dobra wiadomość - bariera wejścia w tworzenie oprogramowania spadła jak nigdy wcześniej.

Jest jednak druga strona medalu. AI, które pisze kod, jest trenowane na miliardach przykładów z internetu - dobrych i złych. Świetnie radzi sobie z tym, żeby coś działało. Znacznie gorzej - z tym, żeby działało bezpiecznie. Modele językowe domyślnie optymalizują pod kątem "czy funkcja spełnia zadanie", a nie "czy ktoś może tę funkcję wykorzystać przeciwko Tobie". W efekcie bardzo łatwo jest dostać działającą aplikację, która jednocześnie ma otwarte drzwi dla kogoś, kto będzie chciał wykraść dane Twoich klientów, przejąć konto administratora albo po prostu zepsuć Ci sklep tuż przed Black Friday.

Nie chodzi o to, żeby się bać albo rezygnować z tworzenia z pomocą AI - to nadal świetne narzędzie. Chodzi o to, żeby wiedzieć, na co zwrócić uwagę, zanim ktoś zwróci na to uwagę za Ciebie - i to w gorszy sposób.

Każdy z 15 punktów tej checklisty opisujemy w trzech krokach:

- **Czym jest mechanizm** - proste wyjaśnienie, o co w ogóle chodzi, bez żargonu (a jeśli fachowy termin się pojawia, tłumaczymy go od razu).
- **Jaki skutek** - co realnie może się stać, jeśli dany element jest źle skonfigurowany lub go brakuje.
- **Jak sprawdzić** - konkretna instrukcja weryfikacji: dla poziomu podstawowego i średniozaawansowanego jest to coś, co zrobisz sam; dla poziomu zaawansowanego - opis metody, którą stosuje osoba z doświadczeniem inżynierskim w bezpieczeństwie.

Poziom podstawowy: sprawdzisz sam w 15 minut

Poniższe pięć punktów zweryfikujesz samodzielnie, z minimalnym zakresem wiedzy technicznej.

1. Klucze i hasła „zaszyte" w kodzie

Czym jest mechanizm: Kiedy tworzysz aplikację z AI, bardzo często proces wygląda tak: AI prosi Cię o "klucz API" (unikalny kod dostępu, który pozwala Twojej aplikacji korzystać z zewnętrznej usługi, np. płatności, wysyłki maili, mapy Google) i wkleja go bezpośrednio w kod, zamiast trzymać go w oddzielnym, ukrytym miejscu.

Jaki skutek: Jeśli taki kod jest publicznie dostępny - np. wrzucony na GitHub (popularny serwis do przechowywania kodu) jako "publiczny", albo widoczny w kodzie źródłowym strony w przeglądarce - każdy może ten klucz skopiować i użyć na Twój koszt albo w Twoim imieniu, np. wysyłając tysiące maili z Twojego konta albo obciążając Twój rachunek za usługę.

Jak sprawdzić: Jeśli trzymasz kod na GitHubie, sprawdź, czy repozytorium (miejsce przechowywania kodu) jest ustawione jako "private" (prywatne), a nie "public". W przeglądarce na swojej stronie kliknij prawym przyciskiem myszy → "Pokaż źródło strony" lub "Zbadaj element" i przeszukaj tekst (Ctrl+F) pod kątem słów takich jak "key", "secret", "password", "token" - jeśli widzisz tam konkretne ciągi znaków przypominające hasła, to sygnał alarmowy.

2. Panel administracyjny pod przewidywalnym adresem

Czym jest mechanizm: Wiele narzędzi AI generuje panel administracyjny (miejsce, z którego zarządzasz treścią, zamówieniami, użytkownikami) pod bardzo oczywistym adresem, np. twojastrona.pl/admin albo /dashboard, często bez żadnego dodatkowego zabezpieczenia poza prostym logowaniem.

Jaki skutek: Przewidywalny adres to pierwszy krok, po którym ktoś niepowołany zaczyna próbować się do panelu dostać - a jeśli logowanie nie ma dodatkowych zabezpieczeń (np. limitu prób), włamanie do panelu administracyjnego oznacza pełną kontrolę nad Twoją aplikacją: danymi klientów, zamówieniami, treścią strony.

Jak sprawdzić: Wpisz w przeglądarce adres swojej strony z dopiskami /admin, /dashboard, /panel, /wp-admin. Jeśli którykolwiek z nich otwiera się bez logowania albo z bardzo prostym formularzem, którego nikt nie zabezpieczył (np. bez limitu prób logowania), to miejsce wymagające pilnej uwagi.

3. Formularze bez podstawowej walidacji danych

Czym jest mechanizm: Walidacja to sprawdzanie, czy dane wpisane przez użytkownika mają sens i odpowiedni format (np. czy pole "e-mail" faktycznie zawiera znak @, czy pole "cena" zawiera liczbę, a nie tekst). AI czasem generuje formularze, które przyjmują dosłownie wszystko, bez żadnej takiej kontroli.

Jaki skutek: Samo w sobie to jeszcze nie jest atak, ale jest pierwszym krokiem do wielu poważniejszych problemów opisanych w dalszych poziomach (np. punkt 11) - brak walidacji to otwarte drzwi, przez które później przechodzą bardziej złożone ataki. Dodatkowo, bez walidacji Twoja baza danych zaczyna zapełniać się "śmieciowymi" wpisami, co utrudnia normalne prowadzenie biznesu.

Jak sprawdzić: Wejdź na swój formularz kontaktowy albo rejestracyjny i spróbuj wpisać w polu "imię" bardzo długi tekst (kilkaset znaków) albo znaki specjalne typu < > " '. Jeśli formularz to przyjmuje bez żadnego komunikatu błędu i zapisuje "jak leci", to znak, że walidacja jest zbyt słaba.

4. Brak certyfikatu SSL (HTTPS)

Czym jest mechanizm: SSL/HTTPS to szyfrowanie połączenia między przeglądarką użytkownika a Twoim serwerem (widoczne w przeglądarce jako kłódka obok adresu strony) - dane przesyłane w formularzach są dzięki temu nieczytelne dla kogokolwiek "podsłuchującego" połączenie po drodze.

Jaki skutek: Bez HTTPS dane przesyłane przez formularze - w tym hasła czy dane karty płatniczej - mogą teoretycznie zostać przechwycone. To też jeden z podstawowych czynników rankingowych Google - strona bez HTTPS jest gorzej pozycjonowana i oznaczana przez przeglądarki jako "niezabezpieczona", co odstrasza klientów już na pierwszy rzut oka.

Jak sprawdzić: Spójrz na pasek adresu w przeglądarce. Powinna być tam ikona kłódki i adres zaczynający się od https://, nie http://. Większość nowoczesnych platform hostingowych (miejsc, gdzie "mieszka" Twoja strona) i narzędzi do vibe-codingu włącza to domyślnie - ale warto to zweryfikować, zwłaszcza jeśli korzystasz z własnej domeny podpiętej ręcznie.

5. Nieaktualizowane biblioteki i zależności

Czym jest mechanizm: Twoja aplikacja prawdopodobnie korzysta z gotowych "klocków" napisanych przez innych programistów (tzw. biblioteki lub pakiety) - np. do obsługi płatności, wysyłki maili czy wyświetlania kalendarza. Te klocki miewają błędy bezpieczeństwa, które są z czasem odkrywane i łatane przez ich twórców.

Jaki skutek: Jeśli AI zbudowało Twoją aplikację raz i od tego czasu nikt jej nie aktualizował, prawdopodobnie korzystasz z wersji z lukami, które są już publicznie znane - a to oznacza, że każdy może je sobie znaleźć w internecie i sprawdzić, czy działają na Twojej stronie.

Jak sprawdzić: Zapytaj osobę, która pomagała Ci przy wdrożeniu (albo samo narzędzie AI, jeśli masz do niego dostęp z historią projektu), kiedy ostatni raz aktualizowano zależności projektu. Jeśli odpowiedź brzmi "nigdy" albo "nie wiem" - to sygnał, żeby zlecić taki przegląd.

Poziom średniozaawansowany

To wymaga już zrozumienia kilku mechanizmów, ale nadal możesz je ogarnąć bez wiedzy programistycznej. Pokazujemy Ci problem i jego realne skutki. Konkretne wdrożenie poprawki to zwykle praca dla kogoś technicznego.

6. Czy widzisz dane innych użytkowników, zmieniając numer w adresie (IDOR)

Czym jest mechanizm: To jedna z najczęstszych i najgroźniejszych luk w aplikacjach budowanych szybko, w tym z pomocą AI. Nazywa się **IDOR** (Insecure Direct Object Reference - niezabezpieczone bezpośrednie odwołanie do obiektu, czyli sytuacja, w której aplikacja pokazuje dane na podstawie samego numeru w adresie, bez sprawdzenia, czy to Twoje dane). Przykład: logujesz się do swojego sklepu, otwierasz swoje zamówienie i widzisz adres w stylu sklep.pl/zamowienie/1042.

Jaki skutek: Jeśli zmienisz ręcznie ten numer na 1043 i zobaczysz zamówienie kogoś innego - łącznie z jego adresem, telefonem, danymi karty czy historią zakupów - to znaczy, że każdy zalogowany użytkownik (a w gorszym wariancie: każdy odwiedzający) może "przeglądać" cudze dane, zamówienia, faktury, a czasem nawet zmieniać cudze ustawienia konta. To jedno z najczęściej wykorzystywanych zjawisk w atakach na aplikacje webowe i regularnie znajduje się w czołówce światowych zestawień najpoważniejszych zagrożeń dla aplikacji internetowych.

Jak sprawdzić: Zaloguj się na dwa różne konta testowe w swojej aplikacji. Na koncie A znajdź adres z numerem (np. zamówienia, faktury, wiadomości) i spróbuj zmienić ten numer na taki, który powinien należeć do konta B. Jeśli zobaczysz dane konta B, będąc zalogowanym jako konto A - to jest realny problem do naprawy.

7. Zarządzanie sesją i wylogowaniem

Czym jest mechanizm: Sesja to "zapamiętanie" przez system, że jesteś zalogowany, żebyś nie musiał podawać hasła przy każdym kliknięciu. Dobrze skonfigurowana sesja ma określony czas ważności i faktycznie się kończy, gdy klikniesz "wyloguj".

Jaki skutek: Problem pojawia się, gdy sesje nie wygasają nigdy (ktoś, kto uzyska dostęp do Twojego urządzenia tydzień później, wciąż będzie zalogowany), albo gdy wylogowanie "na oko" działa, ale w rzeczywistości nie unieważnia dostępu po stronie serwera. Jeśli korzystasz ze wspólnego komputera albo zgubisz telefon, ktoś może mieć dostęp do Twojego konta znacznie dłużej, niż myślisz.

Jak sprawdzić: Zapytaj wprost osobę techniczną (lub w ramach przeglądu bezpieczeństwa), czy sesje mają ustawiony czas wygasania i czy wylogowanie faktycznie unieważnia token (unikalny kod potwierdzający zalogowanie) po stronie serwera, a nie tylko w przeglądarce.

8. Uprawnienia dostępu do bazy danych

Czym jest mechanizm: Baza danych to miejsce, gdzie fizycznie przechowywane są wszystkie informacje - konta, zamówienia, hasła (najczęściej w formie zaszyfrowanej, tzw. "hashowanej"). Dobra zasada bezpieczeństwa to zasada najmniejszych uprawnień: każda część aplikacji powinna mieć dostęp tylko do tych danych, których faktycznie potrzebuje. W praktyce narzędzia AI często konfigurują dostęp "na maksa" - bo tak jest szybciej i "wszystko działa".

Jaki skutek: Jeśli jeden fragment aplikacji zostanie złamany (np. przez lukę opisaną w punkcie 6 albo 11), atakujący automatycznie zyskuje dostęp do znacznie większej ilości danych, niż powinien - bo uprawnienia nie były odpowiednio rozdzielone. Jedna słaba furtka otwiera dostęp do wszystkiego.

Jak sprawdzić: To już temat do rozmowy z osobą techniczną - chodzi o sprawdzenie, czy w bazie danych są skonfigurowane tzw. reguły dostępu na poziomie wiersza (Row Level Security) lub równoważny mechanizm, ograniczający, kto i co może odczytać.

9. Konfiguracja CORS: kto może „rozmawiać" z Twoim backendem

Czym jest mechanizm: CORS (Cross-Origin Resource Sharing - mechanizm kontrolujący, które strony internetowe mogą komunikować się z Twoim serwerem) to ustawienie, które w uproszczeniu mówi: "tylko moja strona ma prawo pytać mój serwer o dane". Bardzo częstym uproszczeniem podczas szybkiego tworzenia (także z pomocą AI) jest ustawienie CORS na "zezwól wszystkim" - bo to eliminuje frustrujące błędy na etapie budowy aplikacji.

Jaki skutek: Przy zbyt luźnej konfiguracji CORS dowolna inna strona internetowa (w tym stworzona specjalnie w złych intencjach) może w niektórych scenariuszach wysyłać zapytania do Twojego serwera, "podszywając się" pod działania zalogowanego użytkownika, co w połączeniu z innymi lukami może prowadzić do kradzieży danych lub wykonywania niechcianych akcji w Twoim imieniu.

Jak sprawdzić: Zapytaj, czy konfiguracja CORS w Twojej aplikacji jest ograniczona wyłącznie do Twojej domeny, a nie ustawiona jako otwarta dla wszystkich ("*").

10. Ochrona przed automatycznymi atakami (rate limiting, captcha)

Czym jest mechanizm: Rate limiting to ograniczenie liczby prób/zapytań, jakie ktoś (albo automatyczny program) może wykonać w danym czasie - np. maksymalnie 5 prób logowania na minutę z jednego adresu. Captcha to z kolei prosty test ("zaznacz obrazki z sygnalizacją świetlną"), który ma odróżnić człowieka od automatycznego programu.

Jaki skutek: Bez limitu prób ktoś może w ciągu kilku godzin "przetestować" tysiące kombinacji haseł na jednym koncie (tzw. atak brute force), albo zasypać Twoją skrzynkę mailową tysiącami automatycznych zgłoszeń z formularza kontaktowego.

Jak sprawdzić: Spróbuj zalogować się na swoje konto testowe z celowo błędnym hasłem 10-15 razy pod rząd w krótkim czasie. Jeśli system w żadnym momencie Cię nie zablokuje ani nie poprosi o captchę, to znak, że tego zabezpieczenia prawdopodobnie brakuje.

Poziom zaawansowany

To już praca wymagająca doświadczenia inżynierskiego w bezpieczeństwie aplikacji. W wersji publikowanej ta sekcja jest skracana do opisu problemu i zaproszenia do kontaktu. Poniżej - pełna treść do użytku wewnętrznego.

11. Testowanie podatności typu SQL Injection i XSS

Czym jest mechanizm: SQL Injection to atak polegający na wstrzyknięciu złośliwego fragmentu zapytania do bazy danych w pole formularza (np. wpisanie w polu "login" specjalnie skonstruowanego tekstu, który sprawia, że baza danych "myśli", że ma zwrócić wszystkie rekordy albo pominąć weryfikację hasła). XSS (Cross-Site Scripting) to z kolei wstrzyknięcie złośliwego kodu, który wykona się w przeglądarce innego użytkownika odwiedzającego stronę.

Jaki skutek: Udany SQL Injection może oznaczać pełny wyciek bazy danych (hasła, dane osobowe, dane kart) albo obejście logowania i przejęcie dowolnego konta. Udany XSS pozwala np. wykraść dane sesji odwiedzającego stronę i przejąć jego konto bez znajomości hasła - np. poprzez komentarz zawierający ukryty skrypt.

Jak sprawdzić (metoda profesjonalna): Rzetelne sprawdzenie odporności aplikacji na te ataki wymaga systematycznego testowania każdego punktu wejścia danych (każdego formularza, parametru w adresie URL, nagłówka zapytania) z użyciem specjalistycznych narzędzi (np. Burp Suite, OWASP ZAP) oraz wiedzy o tym, jak dokładnie skonstruować "złośliwe" dane wejściowe, żeby sprawdzić reakcję systemu, bez uszkadzania produkcyjnych danych. To proces nazywany testem penetracyjnym aplikacji webowej - symulowanym, kontrolowanym atakiem wykonywanym za zgodą właściciela systemu. Standardowa metodyka obejmuje mapowanie wszystkich punktów wejścia aplikacji, testowanie ich w izolowanym środowisku (kopii aplikacji,

nie na produkcji), klasyfikację znalezionych podatności według faktycznego ryzyka i przygotowanie raportu z konkretnymi rekomendacjami naprawczymi wraz z priorytetyzacją.

12. Przegląd kodu pod kątem bezpieczeństwa (SAST)

Czym jest mechanizm: SAST (Static Application Security Testing - statyczna analiza kodu pod kątem bezpieczeństwa) to automatyczne skanowanie całego kodu źródłowego aplikacji w poszukiwaniu wzorców znanych z podatnościami - np. miejsc, gdzie dane od użytkownika trafiają bezpośrednio do zapytania bazodanowego bez odpowiedniego zabezpieczenia (tzw. parametryzacji zapytań), albo gdzie klucze i hasła są zapisane wprost w kodzie.

Jaki skutek: Bez tego etapu w kodzie zostają utrwalone błędy, które AI ma tendencję powielać w wielu miejscach naraz (np. brak walidacji po stronie serwera, poleganie wyłącznie na zabezpieczeniach po stronie przeglądarki, które łatwo ominąć) - a każdy z nich to potencjalna furtka, o której nikt nie wie, dopóki ktoś jej nie wykorzysta.

Jak sprawdzić (metoda profesjonalna): Narzędzia takie jak Semgrep, SonarQube czy Snyk Code potrafią automatycznie przeskanować cały projekt i wskazać setki potencjalnych problemów w kilka minut - ale kluczowa jest tu ludzka weryfikacja wyników (tzw. triage), bo automatyczne skanery generują też sporo fałszywych alarmów. Pełny proces obejmuje: uruchomienie skanera na całym repozytorium, ręczną weryfikację wyników przez osobę z doświadczeniem w bezpieczeństwie aplikacji, priorytetyzację znalezisk według realnego ryzyka biznesowego oraz sprawdzenie konkretnych fragmentów kodu wygenerowanych przez AI pod kątem powtarzalnych, znanych błędów.

13. Zabezpieczenia infrastruktury i hostingu

Czym jest mechanizm: Nawet idealnie napisany kod aplikacji może zostać skompromitowany, jeśli infrastruktura, na której działa (serwer, baza danych, sieć), nie jest odpowiednio zabezpieczona. Obejmuje to m.in. konfigurację WAF (Web Application Firewall - zapory sieciowej filtrującej ruch przychodzący pod kątem znanych wzorców ataków), segmentację sieci (rozdzielenie bazy danych od publicznie dostępnej części aplikacji) oraz konfigurację kopii zapasowych z testowanym procesem odtwarzania.

Jaki skutek: Bez tych elementów nawet drobny błąd w jednym miejscu może dać atakującemu bezpośredni dostęp do bazy danych z poziomu internetu, a sam backup, którego nigdy nie przetestowano pod kątem przywracania, daje fałszywe poczucie bezpieczeństwa - w razie realnego incydentu może się okazać bezużyteczny.

Jak sprawdzić (metoda profesjonalna): Obejmuje to regularne skanowanie infrastruktury pod kątem otwartych portów i nieużywanych usług nasłuchujących, konfigurację odpowiednich nagłówków bezpieczeństwa HTTP (np. Content-Security-Policy, ograniczający, jakie skrypty mogą się wykonać na stronie) oraz - w przypadku popularnych platform niskokodowych/AI (Lovable, Replit, Vercel, Supabase i podobne) - dokładny przegląd ich domyślnych ustawień, bo są one projektowane pod kątem szybkiego uruchomienia, a nie maksymalnego bezpieczeństwa.

14. Bezpieczeństwo łańcucha dostaw (biblioteki i integracje)

Czym jest mechanizm: Twoja aplikacja to nie tylko kod, który napisało dla Ciebie AI - to również dziesiątki, a czasem setki zewnętrznych pakietów i bibliotek, z których ten kod korzysta, oraz integracje z zewnętrznymi usługami (płatności, mailingi, analityka). Każdy z tych elementów to potencjalny wektor ataku, na który nie masz bezpośredniego wpływu.

Jaki skutek: To na tyle poważny i rosnący problem, że w najnowszej, opublikowanej pod koniec 2025 roku aktualizacji światowej listy najpoważniejszych zagrożeń dla aplikacji internetowych (OWASP Top 10 - uznawanego na całym świecie standardowego zestawienia najważniejszych ryzyk bezpieczeństwa aplikacji webowych) po raz pierwszy pojawiła się osobna kategoria dotycząca awarii łańcucha dostaw oprogramowania, która ma najwyższy odnotowany wskaźnik występowania spośród wszystkich kategorii na liście. Oznacza to, że Twoja aplikacja może zostać skompromitowana nie przez błąd w Twoim kodzie, tylko przez lukę w jednej z bibliotek, z których korzysta.

Jak sprawdzić (metoda profesjonalna): Profesjonalne zarządzanie tym ryzykiem obejmuje automatyczne skanowanie zależności projektu pod kątem znanych, opublikowanych luk bezpieczeństwa (np. narzędziami typu Dependabot, Snyk czy npm audit), politykę regularnej aktualizacji pakietów z testowaniem regresyjnym, weryfikację reputacji i utrzymania każdej nowej zewnętrznej biblioteki przed jej dodaniem do projektu, oraz ograniczenie uprawnień, jakie integracje zewnętrzne mają w Twoim systemie, zgodnie z zasadą najmniejszych uprawnień opisaną w punkcie 8.

15. Plan reagowania na incydent bezpieczeństwa

Czym jest mechanizm: Nawet najlepiej zabezpieczona aplikacja może zostać zaatakowana - celem dojrzałego podejścia do bezpieczeństwa nie jest osiągnięcie stanu "zero ryzyka" (to nierealne), tylko przygotowanie się na sytuację, w której coś jednak się wydarzy, tak żeby zminimalizować szkody i czas reakcji.

Jaki skutek: Bez takiego planu włamanie jest zwykle wykrywane dopiero przez klientów, tygodnie po fakcie, a reakcja jest chaotyczna i spóźniona - co pogłębia szkody wizerunkowe i finansowe, a w wielu jurysdykcjach, w tym w Unii Europejskiej na mocy RODO, brak terminowego powiadomienia osób, których dane zostały naruszone, jest dodatkowo prawnym naruszeniem.

Jak sprawdzić (metoda profesjonalna): Profesjonalny plan obejmuje: jasno przypisaną odpowiedzialność (kto podejmuje decyzje w razie wykrycia włamania), gotowy do uruchomienia mechanizm izolacji zagrożonego systemu, skonfigurowane wcześniej logowanie i alerty bezpieczeństwa (żeby w ogóle wiedzieć, że coś się dzieje), procedurę powiadamiania osób, których dane mogły zostać naruszone, oraz proces analizy powłamaniowej (tzw. post-mortem), pozwalający zamknąć tę samą lukę na przyszłość.

Jeśli odhaczyłeś już pierwszych dziesięć punktów tej checklisty - to naprawdę dobra robota. Większość właścicieli aplikacji budowanych z pomocą AI nigdy nie robi nawet tyle.

Pozostałe pięć punktów - testowanie podatności, przegląd kodu pod kątem bezpieczeństwa, zabezpieczenia infrastruktury, ochrona przed lukami w zewnętrznych bibliotekach i plan na wypadek incydentu - to już obszar, w którym trudno się obejść bez kogoś, kto robi to zawodowo. Nie dlatego, że chcemy Cię nastraszyć, tylko dlatego, że te elementy naprawdę wymagają narzędzi i doświadczenia, których nie da się zdobyć w weekend.

W MITS pomagamy właścicielom firm, którzy zbudowali swój produkt cyfrowy samodzielnie lub z pomocą AI, zrozumieć, gdzie realnie stoją pod względem bezpieczeństwa - i co warto zrobić dalej, zanim zainwestują więcej czasu, budżetu marketingowego czy zaufania klientów w produkt, który może mieć ukryte słabe punkty.

Jeśli chcesz wiedzieć, jak wygląda to naprawdę w Twojej aplikacji - **zapraszamy na bezpłatną, wstępną rozmowę**, podczas której spojrzymy razem na Twój produkt i powiemy szczerze, gdzie widzimy ryzyko, a gdzie jest już całkiem nieźle. Bez zobowiązań, bez naciągania na usługi, których nie potrzebujesz - po prostu rzetelna ocena od ludzi, którzy na co dzień budują i zabezpieczają takie aplikacje.

Mits sp. z o.o.

+48 507 972 278 office@mits.pl +48 538 537 623
www.mits.pl