1: Architektura, projektowanie i modelowanie zagrożeń

Cel:

Upewnij się, że weryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:

- Na poziomie 1, elementy aplikacji są zidentyfikowane i mają rację bytu w aplikacji
- Na poziomie 2, architektura jest zdefiniowana, a kod jest zgodny z architekturą
- Na poziomie 3 architektura i design są gotowe, używane, i efektywne

	Opis	1	2	3
1.1	Upewnij się, że wszystkie elementy aplikacji są zidentyfikowane i konieczne.	✓	✓	✓
1.2	Kontrola bezpieczeństwa nigdy nie jest egzekwowana tylko po stronie klienta, ale na odpowiednich zdalnych punktach końcowych.		✓	✓
1.3	Architektura wysokiego poziomu dla aplikacji i wszystkich połączonych usług zdalnych została zdefiniowana, a bezpieczeństwo zostało rozwiązane w tej architekturze.		✓	✓
1.4	Dane uważane za poufne w kontekście aplikacji są jasno określone.			✓
1.5	Wszystkie składniki aplikacji są definiowane pod względem funkcji biznesowych i / lub funkcji bezpieczeństwa, które zapewniają.			✓
1.6	Opracowano model zagrożenia dla aplikacji i powiązanych usług zdalnych, który identyfikuje potencjalne zagrożenia i środki zapobiegawcze..			✓
1.7	Wszystkie kontrole bezpieczeństwa mają scentralizowaną implementację.		✓	✓
1.8	Upewnij się, że elementy są oddzielone od siebie za pomocą określonej kontroli bezpieczeństwa, np segmentacji sieci, reguł zapory lub grup zabezpieczeń opartych na chmurze.		✓	✓
1.9	Istnieje mechanizm wymuszania aktualizacji aplikacji..		✓	✓
1.10	Bezpieczeństwo jest adresowane we wszystkich częściach cyklu rozwoju oprogramowania.		✓	✓
1.11	Wszystkie komponenty aplikacji, biblioteki, moduły, frameworki, platformy i systemy operacyjne są wolne od znanych luk bezpieczeństwa.		✓	✓
1.12	Istnieją wyraźne zasady dotyczące zarządzania kluczami kryptograficznymi (jeśli istnieją) i egzekwowaniem cyklu życia kluczy		✓	✓

	kryptograficznych. Najlepiej postępuj zgodnie ze standardami zarządzania kluczami, takimi jak NIST SP 800-57.			

2: Weryfikacja Wymagań Autoryzacji

Cel:

Uwierzytelnianie jest aktem określenia, lub potwierdzenia, czegoś (lub kogoś) jako autentycznego, to znaczy, sprawdzenia czy dana osoba jest w rzeczywistości tą, za którą się podaje. Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:

● sprawdza tożsamość cyfrową nadawcy komunikatu.

● Zapewnia, że tylko osoby uprawnione mogą uwierzytelniać i dane uwierzytelniania są transportowane w bezpieczny sposób.

	Opis	1	2	3
2.1	Sprawdź, czy wszystkie strony i zasoby wymagają uwierzytelniania po stronie serwera, z wyjątkiem tych, które mają być publiczne.	✓	✓	✓
2.2	Upewnij się, że formularze zawierające dane dostępu nie są wypełniane automatycznie przez aplikacje - ani jako ukryte pola, argumenty URL, requesty Ajax, ani w formularzach. Limity jednorazowe ograniczone czasem są dopuszczalne jako stand in, na przykład w celu ochrony formularzy zmian haseł lub formularzy zapomnianych haseł.	✓	✓	✓
2.6	Zweryfikuj czy nawet jeśli kontrole uwierzytelniania zawiodą, to osoby nieupoważnione nie będą mogły się zalogować.	✓	✓	✓
2.7	Zweryfikuj czy pola wejściowe haseł pozwalają lub zachęcają do używania tekstu szyfrującego i nie zapobiegają wprowadzania haseł, które są długimi lub bardzo złożonymi tekstami szyfrującymi.	✓	✓	✓
2.8	Sprawdź, czy wszystkie funkcje tożsamości (zapomniałem hasła, zmiana hasła, zmień email, zarządzaj tokenem 2FA), mają kontrolę bezpieczeństwa jako podstawowy mechanizm uwierzytelniania.	✓	✓	✓
2.9	Upewnij się, że funkcja zmiany hasła zawiera stare hasło, nowe hasło i potwierdzenie hasła.	✓	✓	✓

2.12	Upewnij się, że wszystkie podejrzane decyzje dotyczące uwierzytelniania są rejestrowane. To powinno zawierać żądania z odpowiednimi metadanymi, potrzebnymi do badań bezpieczeństwa.		✓	✓
2.13	Upewnij się, że hasła kont wykorzystują odpowiednio silną rutynę szyfrowania i że wytrzymują ataki brute force wymierzone w rutynę szyfrowania.		✓	✓
2.16	Sprawdź, czy dane aplikacji są transportowane za pomocą odpowiednio szyfrowanego połączenia(np. TLS).	✓	✓	✓
2.17	Upewnij się, że funkcja 'zapomniane hasło' i inne sposoby odzyskiwania nie ujawniają aktualnego hasła, a nowe hasło nie jest przesyłane do użytkownika w postaci zwykłego tekstu. Zamiast tego należy użyć jednorazowego linku do resetowania hasła.	✓	✓	✓
2.18	Upewnij się, że wyliczenie informacji nie jest możliwe za pomocą loginu, resetowania hasła lub funkcji 'zapomniane konto'.		✓	✓
2.19	Sprawdź, czy nie ma w użyciu domyślnych haseł do frameworku aplikacji lub pozostałych komponentów używanych przez aplikację (na przykład "admin/hasło").	✓	✓	✓
2.20	Upewnij się, że dławienie żądań jest aktywne, aby uniknąć automatycznych ataków, takich jak brute force lub ataków blokady konta.		✓	✓
2.21	Upewnij się, że wszystkie dane uwierzytelniające do usług zewnętrznych dla aplikacji są szyfrowane i przechowywane w bezpiecznym miejscu.		✓	✓
2.22	Sprawdź, czy 'zapomniane hasło' i inne ścieżki odzyskiwania używają TOTP lub innego tokena miękkiego, technologii push, lub mechanizmu odzyskiwania offline. NIST uznał używanie SMS jako przestarzałe i nie powinno być używane.		✓	✓
2.23	Upewnij się, że blokady konta są podzielone na miękkie i twarde, i nie wykluczają się wzajemnie. Jeśli konto jest czasowo zablokowane w sposób miękki ze względu na brute force, to nie powinno to zresetować twardej blokady.		✓	✓
2.24	Upewnij się, że wymagane są pytania oparte na wiedzy użytkownika ("tajne pytania"), pytania te powinny być na tyle silne, aby chronić aplikację, ale nie powinny naruszać praw prywatności.	✓	✓	✓
2.25	Upewnij się, że system może być skonfigurowany tak, aby uniemożliwić korzystanie z konfigurowalnej liczby poprzednich haseł.		✓	✓

2.26	Sprawdź, czy operacje poufne (np. zmiana hasła, zmiana adresu e-mail, itd.) Wymagają ponownego uwierzytelnienia (np. Hasła lub tokena 2FA). Jest to dodatek do CSRF, a nie zamiast tego.		✓	✓
2.27	Upewnij się, że istnieją środki zapobiegające użyciu słabych i często wykorzystywanych haseł.		✓	✓
2.28	Upewnij się, że wszystkie próby uwierzytelnienia, udane lub nieudane, odpowiadają w tym samym średnim czasie odpowiedzi.			✓
2.29	Upewnij się, że sekrety, klucze API i hasła nie są zawarte w kodzie źródłowym, czy internetowych repozytoriach kodu źródłowego.			✓
2.31	Sprawdź, czy użytkownicy mogą rejestrować i używać weryfikacji TOTP, dwuczynnikowej, biometrycznej (Touch ID lub podobnej) lub równoważnego mechanizmu uwierzytelniania wieloskładnikowego, który zapewnia ochronę przed ujawnieniem danych uwierzytelniających pojedynczego czynnika.		✓	✓
2.32	Sprawdź, czy dostęp do interfejsów administracyjnych jest ściśle kontrolowany i nie jest dostępny dla niezaufanych stron.	✓	✓	✓
2.33	Sprawdź, czy aplikacja jest zgodna z menedżerami haseł działającymi w oparciu o przeglądarkę i strony trzecie, chyba, że zabrania tego polityka bezpieczeństwa.	✓	✓	✓

3. Weryfikacja Wymagań Zarządzania Sesją

Cel:

Jednym z najważniejszych elementów każdej aplikacji internetowej jest mechanizm, poprzez który aplikacja kontroluje i zarządza swoim stanem dla użytkownika, który z niej korzysta. Często nazywa się to zarządzaniem sesją i definiuje jako zbiór wszystkich kontroli dotyczących stanowej interakcji między użytkownikiem a aplikacją internetową.

Upewnij się, że weryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu zarządzania sesją:

- Sesje są unikalne dla każdej osoby i nie można ich odgadnąć lub dzielić.
- Sesje są unieważnione, gdy nie są już potrzebne, i upływa ich limit czasu w okresie bezczynności.

	Opis	**1**	**2**	**3**
3.2	Upewnij się, że sesje są unieważnione, gdy użytkownik wyloguje się.	✓	✓	✓

3.3	Upewnij się, że limit czasu sesji upływa po określonym czasie bezczynności.			✓
3.4	Upewnij się, że limit czasu sesji upływa po administracyjnie-konfigurowalnym maksymalnym okresie, niezależnie od aktywności (absolutny timeout).		✓	✓
3.5	Upewnij się, że wszystkie strony, które wymagają uwierzytelnienia mają łatwy i widoczny dostęp do funkcji wylogowania się.	✓	✓	✓
3.6	Upewnij się, że identyfikator sesji nie jest ujawniony w adresach URL, komunikatach o błędach lub logach. Obejmuje to sprawdzenie, że aplikacja nie obsługuje przepisywania URL ciasteczek sesji.			✓
3.7	Upewnij się, że wszystkie pomyślne uwierzytelnienia i ponowne uwierzytelnienia generują nowy identyfikator sesji.	✓	✓	✓
3.10	Sprawdź, czy tylko identyfikatory sesji wygenerowane przez strukturę aplikacji są rozpoznawane jako aktywne przez aplikację.		✓	✓
3.11	Testuj identyfikatory sesji na podstawie takich kryteriów, jak losowość, unikalność, odporność na analizę statystyczną i kryptograficzną oraz wyciek informacji.	✓	✓	✓
3.12	Sprawdź, czy identyfikatory sesji przechowywane w plikach cookie mają zasięg na podstawie atrybutu "ścieżka"; i włączają flagi cookie "HttpOnly" i "Secure".	✓	✓	✓
3.17	Sprawdź, czy aplikacja śledzi wszystkie aktywne sesje i pozwala użytkownikom na zakończenie sesji w sposób wybiórczy lub globalny ze swojego konta.		✓	✓
3.18	Sprawdź, czy dla aplikacji o wysokiej wartości użytkownik jest monitowany o możliwość zakończenia wszystkich innych aktywnych sesji po pomyślnym zakończeniu procesu zmiany hasła.			✓
3.1	Zostanie określone w późniejszym terminie.	✓	✓	✓

4. Weryfikacja Kontroli dostępu

Cel:

Autoryzacja umożliwia dostęp do zasobów tym, którzy mają na to pozwolenie. Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:

- Osoba korzystająca z zasobów posiada ważne dane uwierzytelniające.
- Użytkownicy są związani dobrze zdefiniowanym zestawem ról i uprawnień.
- Metadane ról i uprawnień są chronione przed powtarzaniem lub sabotażem.

	Opis	1	2	3
4.1	Upewnij się, że zasada najmniejszego uprzywilejowania ma zastosowanie - użytkownicy powinni być tylko w stanie korzystać z funkcji, plików danych, adresów URL, sterowników, usług i innych zasobów, dla których posiadają specjalne uprawnienie. Oznacza to ochronę przed spoofingiem i podniesienie uprawnień.	✓	✓	✓
4.4	Upewnij się, że dostęp do poufnych zapisów jest chroniony, tak, że tylko autoryzowane obiekty lub dane są dostępne dla każdego użytkownika (na przykład po to by chronić przed użytkownikami manipulującymi parametrem, aby zobaczyć lub zmienić konto innego użytkownika).	✓	✓	✓
4.5	Sprawdź czy przeglądanie katalogów jest wyłączone, chyba że jest to zamierzone działanie. Dodatkowo, aplikacje nie powinny umożliwiać odkrycia lub ujawnienia metadanych pliku lub katalogu, takich jak Thumbs.db, .DS_Store, .git lub folderów .svn.	✓	✓	✓
4.8	Upewnij się, że nawet w przypadku, gdy kontrola dostępu zawiedzie, odbywa się to w sposób bezpieczny.	✓	✓	✓
4.9	Upewnij się, że te same zasady kontroli dostępu implikowane przez warstwę prezentacji są wykonywane po stronie serwera.	✓	✓	✓
4.10	Upewnij się, że wszystkie atrybuty użytkowników i danych, oraz informacje używane przez kontrolę dostępu nie mogą być manipulowane przez użytkowników końcowych, bez specjalnego upoważnienia.		✓	✓
4.11	Upewnij się, że istnieje scentralizowany mechanizm (w tym dla bibliotek zewnętrznych wywołujących zewnętrzne usługi autoryzacji) do ochrony dostępu do każdego rodzaju chronionych zasobów.			✓
4.12	Upewnij się, że wszystkie decyzje dotyczące kontroli dostępu mogą być rejestrowane a wszystkie nieudane decyzje są rejestrowane.		✓	✓
4.13	Upewnij się, że aplikacja lub framework korzysta z silnych losowych tokenów przeciwko atakom CSRF, lub ma inny mechanizm ochrony transakcji.	✓	✓	✓
4.14	Sprawdź, czy system może chronić przed zbiorczym lub ciągłym dostępem do zabezpieczonych funkcji, zasobów lub danych. Na przykład, należy rozważyć użycie regulatora zasobów, aby ograniczyć liczbę edycji na godzinę, aby zapobiec wyczyszczeniu całej bazy danych przez indywidualnego użytkownika.		✓	✓

4.15	Sprawdź, czy aplikacja ma dodatkowe zezwolenia (takie jak uwierzytelnianie step up lub adaptacyjne) dla systemów o niskiej wartości i/lub podział obowiązków dla aplikacji o wysokiej wartości, do egzekwowania kontroli zwalczania nadużyć finansowych.		✓	✓
4.16	Upewnij się, że aplikacja poprawnie egzekwuje autoryzacje kontekstowe, tak aby nie umożliwiać nieautoryzowanego manipulowania za pomocą manipulacji parametrem.	✓	✓	✓

5. Weryfikacja Obsługi Złośliwych Danych Wejściowych

Cel:

Najbardziej powszechnym słabym punktem bezpieczeństwa aplikacji internetowych jest brak właściwej walidacji poprawności danych wejściowych, pochodzących od klienta lub środowiska, przed wykorzystaniem tych danych. To uchybienie prowadzi do niemal wszystkich najważniejszych luk w zabezpieczeniach aplikacji internetowych, takich jak Cross Site Scripting, SQL injection, ataki locale/Unicode, ataki na system plików i przepełnienie bufora.
Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:
● Wszystkie dane wejściowe są weryfikowane pod kątem prawidłowości i dopasowania do zamierzonego celu.
● Nie powinno się nigdy ufać danym pochodzącym od podmiotów zewnętrznych lub klientów, i powinno się takie dane odpowiednio traktować.

	Opis	**1**	**2**	**3**
5.3	Sprawdź, czy błędy sprawdzania poprawności danych wejściowych po stronie serwera powodują odrzucenie żądania i są rejestrowane.	✓	✓	✓
5.5	Sprawdź, czy procedury sprawdzania poprawności danych wejściowych są wymuszane po stronie serwera.	✓	✓	✓
5.6	Sprawdź, czy procedury walidacji danych wejściowych są egzekwowane po stronie serwera.	✓	✓	✓
5.10	Sprawdź, czy wszystkie zapytania do bazy danych są chronione za pomocą sparametryzowanych zapytań lub prawidłowego użycia ORM, aby uniknąć wstrzyknięcia SQL.	✓	✓	✓
5.11	Sprawdź, czy aplikacja jest podatna na wstrzyknięcie LDAP, lub czy kontrole bezpieczeństwa zapobiegają wstrzyknięciu LDAP.	✓	✓	✓

5.12	Upewnij się, że aplikacja nie jest podatna na OS Command injection, lub że kontrole bezpieczeństwa temu zapobiegają.	✓	✓	✓
5.13	Upewnij się, że aplikacja nie jest podatna na wprowadzenie pliku zdalnego (RFI) lub na integrację pliku lokalnego (LFI), gdy używana treść jest ścieżką pliku.	✓	✓	✓
5.14	Upewnij się, że aplikacja nie jest podatna na powszechne ataki XML, takich jak manipulacja zapytań XPath, XML External Entity oraz wstrzyknięcia XML.	✓	✓	✓
5.15	Upewnij się, że wszystkie zmienne typu string umieszczone w kodzie HTML, lub innym kodzie klienta web, są prawidłowo kontekstowo zakodowane ręcznie, lub wykorzystują szablony, które automatycznie kodują kontekstowo, by zapewnić, że aplikacja nie jest podatna na ataki odbite, przechowywane i DOM Cross-Site Scripting (XSS).	✓	✓	✓
5.16	Sprawdź, czy aplikacja nie zawiera błędów związanych z przypisaniem parametrów masowych (tzw. automatycznym wiązaniem zmiennych).		✓	✓
5.17	Sprawdź, czy aplikacja ma zabezpieczenia przed atakami zanieczyszczeń parametrów HTTP, szczególnie jeśli framework aplikacji nie rozróżnia źródeł parametrów żądań (GET, POST, ciasteczka, nagłówki, środowisko, itp)		✓	✓
5.19	Upewnij się, że wszystkie dane wejściowe są zweryfikowane, nie tylko pola formularza HTML, ale wszystkie źródła wejścia (takie jak wywołanie REST, parametry zapytania, nagłówki HTTP, ciasteczka, pliki wsadowe, kanały RSS, itp) za pomocą pozytywnej walidacji (białej listy), a następnie mniejsze formy walidacji takie jak greylisting (szare listy) lub odrzucanie złych wejść (czarna lista).		✓	✓
5.20	Upewnij się, że ustrukturyzowane dane są oznaczone i sprawdzane według określonego schematu, zawierającego dozwolone znaki, długości i strukturę (np numey kart kredytowych, numer telefonu; lub walidację, że powiązanie pól jest uzasadnione, np. że kod pocztowy odpowiada dzielnicy).		✓	✓
5.21	Sprawdź, czy niestrukturalne dane są oczyszczane, by egzekwować ogólne środki bezpieczeństwa, takie jak dozwolone znaki i długości, a znaków potencjalnie szkodliwych w danym kontekście należy unikać (np. nazwiska zawierające Unicode lub apostrofy, jak ね こ lub O'Hara)		✓	✓
5.22	Upewnij się, że niezaufane HTML pochodzące od edytorów WYSIWYG, lub im podobne, są odpowiednio oczyszczane i	✓	✓	✓

	obsługiwane w odpowiedni sposób, według zadania sprawdzania poprawności danych wejściowych i zadania kodowania.			
5.24	Sprawdź, czy w przypadku przenoszenia danych z jednego kontekstu DOM na inny, transfer korzysta z bezpiecznych metod JavaScript, takich jak użycie innerText lub .val, aby upewnić się, że aplikacja nie jest podatna na ataki typu DOM-Cross-Site Scripting (XSS).		✓	✓
5.25	Zweryfikuj, czy podczas analizowania JSON w przeglądarkach lub w backend-ach opartych na JavaScript JSON.parse jest używany do parsowania dokumentu JSON. Nie używaj eval () do parsowania JSON.		✓	✓
5.27	Zweryfikuj aplikację pod kątem luk Server Side Request Forgery	✓	✓	✓
5.28	Sprawdź, czy aplikacja poprawnie ogranicza parsery XML do używania tylko najbardziej restrykcyjnej konfiguracji i do zapewnienia, że niebezpieczne funkcje, takie jak rozwiązywanie zewnętrznych elementów, są wyłączone.	✓	✓	✓
5.29	Sprawdź, czy unika się deserializacji niezaufanych danych lub czy jest ona dogłębnie chroniona, gdy nie można uniknąć deserializacji.	✓	✓	✓

7. Kryptografia

Cel:

Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:

● Nawet gdy moduły kryptograficzne zawiodą, odbywa się to w bezpieczny sposób, a błędy są prawidłowo obsługiwane.

● Zastosowany jest odpowiedni generator liczb losowych, gdy wymagana jest losowość.

● Dostęp do kluczy jest zarządzany w sposób bezpieczny.

	Opis	**1**	**2**	**3**
7.2	Upewnij się, że nawet jeśli moduły kryptograficzne zawiodą, odbywa się to w sposób bezpieczny, a błędy są traktowane w sposób, który nie pozwala na oracle padding.	✓	✓	✓
7.6	Upewnij się, że wszystkie liczby losowe, przypadkowe nazwy plików, losowe GUID i ciągi są generowane przy użyciu zatwierdzonego generatora liczb losowych modułów kryptograficznych, gdy te wartości mają być nie do odgadnięcia przez atakującego.		✓	✓

7.7	Sprawdź, czy algorytmy kryptograficzne używane przez aplikację zostały zatwierdzone według FIPS 140-2 lub równoważnego standardu.	✓	✓	✓
7.8	Sprawdź, czy moduły kryptograficzne działają w ich zatwierdzonym trybie według ich polityki bezpieczeństwa.			✓
7.9	Upewnij się, że istnieje wyraźna polityka zarządzania kluczami kryptograficznymi (np. jak są generowane, rozprowadzane, odwoływane i wygaszane). Upewnij się, że cykl tego klucza jest właściwie egzekwowany.		✓	✓
7.11	Upewnij się, że żaden konsument usług kryptograficznych nie ma bezpośredniego dostępu do materiałów kluczowych. Odizolouj procesy kryptograficzne, w tym tajemnice i rozważ zastosowanie klucza sprzętowego (HSM).			✓
7.12	Upewnij się, że dane osobowe umożliwiające identyfikację osób (PII) i inne poufne dane są przechowywane w postaci zaszyfrowanej w spoczynku.		✓	✓
7.13	Upewnij się, że poufne hasła lub materiały kluczowe przechowywane w pamięci są nadpisywane zerami, gdy tylko nie są już potrzebne, w celu złagodzenia ataków polegających na zrzucaniu pamięci.		✓	✓
7.14	Upewnij się, że wszystkie klucze i hasła mogą być zmienione i są generowane lub zmieniane w czasie instalacji.		✓	✓
7.15	Upewnij się, że liczby losowe są tworzone z przy użyciu odpowiedniej entropii, nawet gdy aplikacja jest pod dużym obciążeniem, albo że aplikacja obniża pułap w takich okolicznościach.			✓

8. Weryfikacja Obsługi i Rejestrowania Błędów

Cel:

Głównym celem obsługi błędów i logowania jest dostarczenie przydatnych reakcji użytkownika, administratorów i zespołów reagowania na incydenty. Celem nie jest stworzenie ogromnej ilości logów, ale logów wysokiej jakości.

Dzienniki wysokiej jakości często zawierają poufne dane, i muszą być chronione zgodnie z lokalnymi przepisami dotyczącymi ochrony danych i dyrektyw. Powinno to obejmować:

- Niezbieranie i niezapisywanie poufnych informacji jeśli nie są one wymagane.
- Zapewnienie, że wszystkie zapisane informacje są przetwarzane w bezpieczny sposób i chronione zgodnie z ich klasyfikacją.
- Zapewnienie, że logi nie są przechowywane w nieskończoność, ale mają możliwie jak najkrótszą żywotność.

Jeśli logi zawierają dane prywatne lub poufne, których definicja różni się w zależności od kraju, stają się one jednymi z najbardziej wrażliwych informacji posiadanych przez aplikację, dzięki czemu są bardzo atrakcyjne dla hakerów.

	Opis	**1**	**2**	**3**
8.1	Upewnij się, że aplikacja nie pokazuje komunikatów o błędach lub śladów stosów (??) zawierających poufne dane, takie jak identyfikator sesji, wersja oprogramowania/frameworku i danych osobowych, które mogą pomóc hakerom.	✓	✓	✓
8.2	Upewnij się, że logika obsługi błędów w kontroli bezpieczeństwa zabrania dostępu domyślnego.		✓	✓
8.3	Sprawdź, czy kontrole rejestracji zabezpieczeń zapewniają możliwość zapisywania udanych, a szczególnie zdarzeń zakończonych niepowodzeniem, które są uznane za istotne z punktu widzenia bezpieczeństwa.		✓	✓
8.4	Upewnij się, że każde zdarzenie logu zawiera niezbędne informacje, które pozwoliłyby na szczegółowe przeanalizowania osi czasu.		✓	✓
8.5	Upewnij się, że wszystkie wydarzenia, które obejmują niezaufane dane, nie będą wykonane jako kod w docelowym oprogramowaniu do przeglądania logów.		✓	✓
8.6	Upewnij się, że logi bezpieczeństwa są chronione przed nieautoryzowanym dostępem i modyfikacją.		✓	✓
8.7	Upewnij się, że aplikacja nie rejestruje danych poufnych w rozumieniu przepisów lokalnego prawa prywatności, danych poufnych organizacji, jakie określono na podstawie oceny ryzyka, lub poufnych danych uwierzytelniających, w tym identyfikatorów sesji użytkownika, haseł, hashów, lub tokenów API, które mogłyby pomóc hakerom.		✓	✓
8.8	Upewnij się, że wszystkie symbole niemożliwe do wydrukowania i separatory pól są odpowiednio zakodowane we wpisach logów, aby zapobiec wstrzyknięciom logów.			✓
8.9	Upewnij się, że pola logów z niezaufanych i zaufanych źródeł można rozróżnić we wpisach logów.			✓
8.10	Upewnij się, że dziennik audytu (lub podobna funkcja) umożliwia niezaprzeczalność kluczowych transakcji.	✓	✓	✓
8.11	Upewnij się, że logi bezpieczeństwa mają jakąś formę sprawdzenia			✓

	integralności lub kontrole w celu zapobiegania nieautoryzowanym zmianom.			
8.12	Upewnij się, że logi są przechowywane na innej partycji niż działająca aplikacja z właściwą rotacją logów.			✓
8.13	“Źródła czasu” powinny być synchronizowane z odpowiednią strefą czasową i godziną.	✓	✓	✓

9. Weryfikacja Ochrony Danych

Cel:
Istnieją trzy kluczowe elementy należytej ochrony danych: poufność, integralność i dostępności (Confidentiality, Integrity and Availability - CIA). Norma ta zakłada, że ochrona danych jest egzekwowana na zaufanym systemie, jak na przykład serwer, który został wzmocniony i ma wystarczające zabezpieczenia. Aplikacja musi zakładać, że wszystkie urządzenia użytkownika są zagrożone w jakiś sposób. W przypadku gdy aplikacja przesyła lub przechowuje poufne informacje na temat niezabezpieczonych urządzeń, takich jak dzielone komputery, telefony i tablety, aplikacja jest odpowiedzialna za zapewnienie że dane przechowywane na tych urządzeniach są szyfrowane i nie mogą być łatwo nielegalnie uzyskane, zmienione lub ujawnione.
Upewnij się, że zweryfikowana aplikacja spełnia następujące wymogi:
● Poufność: Dane powinny być chronione przed nieuprawnionym ujawnieniem lub obserwacją, zarówno w czasie przesyłania i po zapisywaniu.
● Integralność: dane powinny być chronione przed złośliwym tworzeniem, zmienianiem lub usunięciem.
● Dostępność: dane powinny być dostępne dla upoważnionych użytkowników zgodnie z wymaganiami.

	Opis	**1**	**2**	**3**
9.1	Upewnij się, że wszystkie formy zawierające poufne informacje mają wyłączone buforowanie po stronie klienta, włącznie z funkcjami autouzupełniania.	✓	✓	✓
9.2	Upewnij się, że lista poufnych danych przetwarzanych przez aplikację jest zidentyfikowana i że istnieje wyraźna polityka kontroli dostępu, szyfrowania i egzekwowania, zgodna z odpowiednimi dyrektywami dotyczącymi ochrony danych.			✓
9.3	Upewnij się, że wszystkie poufne dane są przesyłane do serwera w treści wiadomości lub nagłówków HTTP (parametry URL nigdy nie są	✓	✓	✓

	używane do przesyłania poufnych danych).			
9.4	Sprawdź, czy aplikacja ustawia odpowiednie nagłówki zapobiegające buforowaniu w taki sposób, że wszelkie poufne i osobiste informacje wyświetlane przez aplikację lub wprowadzane przez użytkownika nie powinny być buforowane na dysku przez popularne współczesne przeglądarki (np. Odwiedź about:cache, aby przejrzeć pamięć podręczną dysku).	✓	✓	✓
9.5	Upewnij się, że wszystkie buforowania lub tymczasowe kopie poufnych danych przechowywane na serwerze, są chronione przed nieupoważnionym dostępem lub czyszczone/unieważniane, po tym jak autoryzowany użytkownik uzyskuje dostęp do poufnych danych.		✓	✓
9.6	Upewnij się, że istnieje sposób, aby usunąć każdego rodzaju poufne dane z aplikacji, na końcu wymaganej polityki przechowywania.			✓
9.7	Sprawdź, czy aplikacja minimalizuje liczbę parametrów żądania, takich jak ukryte pola, zmienne Ajax, ciasteczka i wartości nagłówka.		✓	✓
9.8	Sprawdź, czy aplikacja potrafi wykryć i ostrzegać o nieprawidłowych liczbach żądań gromadzenia danych.			✓
9.9	Upewnij się, że dane przechowywane po stronie klienta - takie jak lokalny magazyn HTML5, przechowywanie sesji, IndexedDB, regularne pliki cookie lub ciasteczkaFlash - nie zawierają poufnych informacji lub PII.	✓	✓	✓
9.10	Sprawdź, czy dostęp do danych poufnych jest rejestrowany, jeśli dane są zbierane zgodnie z odpowiednimi dyrektywami o ochronie danych, lub gdy wymagane jest rejestrowanie dostępów.		✓	✓
9.11	Sprawdź, czy poufne informacje przechowywane w pamięci są nadpisywane zerami, gdy tylko nie są już potrzebne, aby złagodzić ataki polegające na zrzuceniu pamięci.		✓	✓
9.12	Symbol zastępczy (ang. placeholder) dla GDPL	✓	✓	✓
9.13	Sprawdź, czy dane uwierzytelnione są usuwane z pamięci klienta, na przykład DOM przeglądarki, po zakończeniu sesji klienta.		✓	✓

10. Weryfikacja Komunikacji

Cel:
Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:
● TLS jest używany, gdy poufne dane są wysyłane.
● Silne algorytmy i szyfry są stosowane w każdej chwili.

	Opis	1	2	3
10.1	Upewnij się, że ścieżka może być zbudowana z zaufanego CA do każdego TLS certyfikatu serwera, i że każdy certyfikat serwera jest prawidłowy.	✓	✓	✓
10.2	Sprawdź czy TLS jest używany dla wszystkich połączeń (w tym zarówno połączeń zewnętrznych i backend), które są uwierzytelnione lub które dotyczą poufnych danych lub funkcji, i nie używa on protokołów niezabezpieczonych lub niekodowanych. Upewnij się, że najsilniejszą alternatywą jest algorytm preferowany.	✓	✓	✓
10.3	Sprawdź, czy nieudane połączenia z backend TLS są rejestrowane.			✓
10.4	Sprawdź, czy ścieżki certyfikatów są budowane i weryfikowane dla wszystkich certyfikatów klienta za pomocą skonfigurowanych zaufanych kotwic i unieważnienia informacji.			✓
10.5	Upewnij się, że wszystkie połączenia do zewnętrznych systemów, które dotyczą poufnych informacji lub funkcji są uwierzytelnione.		✓	✓
10.6	Upewnij się, że jest jeden standard wdrożenia TLS, który jest używany przez aplikację, która jest skonfigurowana do pracy w zatwierdzonym trybie pracy.			✓
10.7	Sprawdź, czy przypinanie kluczy publicznych certyfikatu TLS jest realizowane za pomocą kluczy produkcji i kopii zapasowych kluczy publicznych.		✓	✓
10.8	Sprawdź, czy nagłówki HTTP Strict Transport Security są zawarte we wszystkich żądaniach i wszystkich subdomenach, takich jak Strict-Transport-Security: max-age=15724800; includeSubdomains	✓	✓	✓
10.9	Upewnij się, że adres URL produkcyjnej strony internetowej został zamieszczony na liście domen Strict Transport Security, prowadzonej przez producentów przeglądarek internetowych.			✓
10.10	Upewnij się, że funkcja Perfect Forward Secrecy jest skonfigurowana, w celu ograniczenia rejestrowania ruchu przez hakerów.	✓	✓	✓

10.11	Upewnij się, że jest włączone i skonfigurowane cofnięcie certyfikacji, takie jak Online Certificate Status Protocol (OSCP) Stapling.	✓	✓	✓
10.12	Upewnij się, że tylko silne algorytmy, szyfry i protokoły są używane przez wszystkie certyfikaty, w tym certyfikaty root i pośrednie wybranej instytucji certyfikującej.	✓	✓	✓
10.13	Sprawdź, czy ustawienia TLS są zgodne z aktualnymi wiodącymi praktykami, zwłaszcza, że typowe konfiguracje, szyfry i algorytmy stają się niebezpieczne.	✓	✓	✓

13. Złośliwy Kod

Cel:

Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania na wysokim poziomie:

● wykryte szkodliwe działania są obsługiwane bezpiecznie i właściwie, tak by nie wpływały na resztę aplikacji.

● nie ma wbudowanych “bomb zegarowych” lub innych ataków opartych na czasie.

● nie "dzwoni do domu" do złośliwych lub nieautoryzowanych miejsc docelowych.

● aplikacja nie ma backdorów, Easter Eggs, ataków salami ani błędów logicznych, które mogą być kontrolowane przez atakującego

Złośliwy kod jest niezwykle rzadki i jest trudny do wykrycia. Ręczny przegląd kodu linia po linii może pomóc w szukaniu bomb logicznych, ale nawet najbardziej doświadczony recenzent kodu będzie miał trudności ze znalezieniem złośliwego kodu, nawet jeśli wie o istnieniu. Ta sekcja nie może zostać ukończona bez dostępu do kodu źródłowego, w tym do jak największej liczby bibliotek stron trzecich.

	Opis	**1**	**2**	**3**
13.1	Sprawdź, czy wszystkie szkodliwe działania są odpowiednio przenoszone do piaskownicy (sandboxed), kontenerowane lub izolowane by opóźnić i powstrzymać hakerów przed atakami na inne aplikacje.	✓	✓	✓
13.2	Upewnij się, że kod źródłowy i tyle bibliotek, ile to możliwe nie zawiera backdorów, easter eggs i wad logicznych w uwierzytelnianiu, kontroli dostępu, danych wejściowych i w logice biznesowej transakcji o dużym znaczeniu.	✓	✓	✓

15. Weryfikacja Logiki Biznesowej

Cel:
Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:
● Przepływ logiki biznesowej jest sekwencyjny i uporządkowany.
● Logika biznesowa obejmuje limity wykrywania i zapobiegania atakom automatycznym, takie jak ciągłe przekazywanie niewielkich funduszy lub dodanie miliona znajomych za jednym razem itd.
● Przepływy logiki biznesowej o wysokiej wartości uwzględniają przypadki nadużyć i złośliwych aktorów oraz ochronę przed podszywaniem się, fałszowaniem, odrzucaniem, ujawnianiem informacji i atakami z podniesionymi uprawnieniami.

	Opis	1	2	3
15.1	Sprawdź, czy aplikacja będzie tylko przetwarzać przepływ logiki biznesowej w kolejności sekwencyjnej, wszystkie etapy będą przetwarzane w realistycznym czasie ludzkim, w ustalonej kolejności, bez pomijania kroków, przetwarzania etapów innego użytkownika lub zbyt szybko zgłoszonych transakcji.		✓	✓
15.2	Sprawdź, czy aplikacja ma ograniczenia biznesowe i poprawnie egzekwuje je od każdego użytkownika, z konfigurowalnymi ostrzeżeniami i automatycznymi reakcjami na zautomatyzowany lub nietypowy atak.		✓	✓

16. Weryfikacja plików i zasobów

Cel:
Upewnij się, że zweryfikowana aplikacja spełnia następujące wymagania wysokiego poziomu:
● niezaufane pliki danych należy odpowiednio i w bezpieczny sposób obsługiwać.
● pliki uzyskane z niezaufanych źródeł są przechowywane poza Webroot i mają ograniczone uprawnienia

	Opis	1	2	3
16.1	Sprawdź, czy przekierowania i przekazania adresów URL pozwalają tylko na miejsca docelowe wyszczególnione na whitelistach, lub pokazują ostrzeżenie w przypadku przekierowania do potencjalnie niezaufanych treści.	✓	✓	✓

16.2	Sprawdź, czy niezaufane dane pliku dodane do aplikacji nie są używane bezpośrednio z poleceniami I/O pliku, zwłaszcza w celu ochrony przed przechodzeniem ścieżki (path traversal), inkludowaniem pliku lokalnego, plikami typu MIME i lukami OS wtryskowych poleceń.	✓	✓	✓
16.3	Upewnij się, że pliki uzyskane z niezaufanych źródeł są sprawdzane pod kątem oczekiwanego typu i skanowane przez skanery antywirusowe, aby zapobiec uploadowi znanej szkodliwej zawartości	✓	✓	✓
16.4	Sprawdź, czy niezaufane dane nie są wykorzystywane w ramach integracji, loadera klas, lub funkcji odbicia, aby zapobiec podatności na zdalne/lokalne wykonanie kodu.	✓	✓	✓
16.5	Sprawdź, czy niezaufane dane nie są wykorzystywane w ramach podziału zasobów między domenami (CORS) w celu ochrony przed niepożądaną zawartością zdalną.	✓	✓	✓
16.6	Upewnij się, że pliki uzyskane z niezaufanych źródeł są przechowywane poza webroot, z ograniczonymi uprawnieniami, najlepiej z silną walidacją.		✓	✓
16.7	Upewnij się, że serwer web lub serwer aplikacji jest domyślnie skonfigurowany do odmowy dostępu do zdalnych zasobów lub systemów poza serwerem web lub serwerem aplikacji.		✓	✓
16.8	Sprawdź kod aplikacji nie wykonuje przesłanych danych otrzymanych z niezaufanych źródeł.	✓	✓	✓
16.9	Upewnij się, że nie są używane nieobsługiwane, niezabezpieczone lub przestarzałe technologie po stronie klienta, takie jak wtyczki NSAPI, Flash, Shockwave, Active-X, Silverlight, NACL, Javy po stronie klienta.	✓	✓	✓
16.10	Sprawdź, czy nagłówek Access-Control-Allow-Origin podziału zasobów między domenami (CORS) nie odzwierciedla po prostu nagłówka pochodzenia żądania lub wspiera pochodzenie "null".	✓	✓	✓

17. Weryfikacja wersji mobilnej

W trakcie usuwania

Ta sekcja poprzednio zawierała regulacje specyficzne dla aplikacji mobilnej, ale obecnie jest usuwana i zastępowana przez OWASP Mobile Application Security Verification Standard.

18. API

Cel:
Upewnij się, że zweryfikowana aplikacja, która używa usługi internetowe oparte na REST lub SOAP ma:
● Odpowiednie uwierzytelnianie, zarządzanie sesjami i autoryzację wszystkich usług internetowych
● Walidację danych wejściowych wszystkich parametrów, które transportuje z niższego do wyższego poziomu zaufania
● Podstawową interoperacyjność warstwy usług internetowych SOAP by promować użycie API

	Opis	1	2	3
18.1	Upewnij się, że ten sam styl kodowania jest używany między klientem a serwerem.	✓	✓	✓
18.2	Upewnij się, że dostęp do funkcji administracji i zarządzania w ramach Web Application Service jest ograniczony do administratorów usług internetowych.	✓	✓	✓
18.3	Sprawdź, czy schemat XML lub JSON został wdrożony i zweryfikowany przed zaakceptowaniem danych wejściowych.	✓	✓	✓
18.4	Upewnij się, że wszystkie dane wejściowe są ograniczone do odpowiedniego rozmiaru.	✓	✓	✓
18.5	Upewnij się, że usługi sieciowe oparte na SOAP są zgodne co najmniej z Web Services-Interoperability (WS-I) Basic Profile. To zasadniczo oznacza szyfrowanie TLS.	✓	✓	✓
18.7	Sprawdź, czy usługa REST jest chroniona przed CSRF poprzez korzystanie z przynajmniej jednej z następujących rzeczy: sprawdzanie nagłówków żądania ORGIN, podwójne przedkładanie ciasteczek, niezgodności(?) CSRF czy sprawdzanie nagłówków żądań odsyłaczy.	✓	✓	✓
18.8	Sprawdź, czy usługa REST bezpośrednio sprawdza przychodzący Content-Type pod kątem oczekiwanego typu, jak np application/xml lub application/json.		✓	✓
18.9	Upewnij się, że treść komunikatu jest podpisana, aby zapewnić niezawodny transport między klientem a usługą poprzez JSON Web Signing lub WS-Security dla żądań SOAP.		✓	✓
18.10	Upewnij się, że alternatywne i mniej bezpieczne ścieżki dostępu nie istnieją.		✓	✓

19. Weryfikacja Konfiguracji

Cel:
Upewnij się, że zweryfikowana aplikacja ma:
● Aktualne biblioteki i platformy.
● Domyślnie bezpieczną konfigurację.
● Wystarczająca wzmocnienia, że zainicjowane przez użytkownika zmiany w domyślnej konfiguracji nie narażają bezpieczeństwa i nie tworzą luk w zabezpieczeniach lub wad systemów bazowych.

	Opis	1	2	3
19.1	Sprawdź, czy wszystkie komponenty są aktualne z właściwą konfiguracją bezpieczeństwa i wersjami. Powinno to obejmować usunięcie niepotrzebnych konfiguracji i folderów, takich jak przykładowe aplikacje, dokumentacja platformy oraz domyślni lub przykładowi użytkownicy.	✓	✓	✓
19.2	Sprawdź, czy komunikacja między komponentami, na przykład między serwerem aplikacji a serwerem bazy danych, jest szyfrowana, szczególnie gdy komponenty znajdują się w różnych “pojemnikach” lub w różnych systemach.		✓	✓
19.3	Sprawdź, czy komunikacja między komponentami, na przykład między serwerem aplikacji i serwerem bazy danych, jest uwierzytelniana przy użyciu konta z najmniej koniecznymi uprawnieniami.		✓	✓
19.4	Sprawdź, czy wdrożenia aplikacji są odpowiednio sandbox-owane, konteneryzowane lub izolowane w celu opóźnienia i powstrzymania atakujących przed atakowaniem innych aplikacji.		✓	✓
19.5	Sprawdź, czy procesy budowania i wdrażania aplikacji są wykonywane w bezpiecznej i powtarzalnej metodzie, takiej jak automatyzacja CI / CD i zautomatyzowane zarządzanie konfiguracją.		✓	✓
19.6	Sprawdź, czy upoważnieni administratorzy mają możliwość zweryfikowania integralności wszystkich konfiguracji istotnych dla bezpieczeństwa, aby wykryć sabotaż.			✓
19.7	Upewnij się, że wszystkie elementy aplikacji są podpisane.			✓
19.8	Sprawdź, czy komponenty zewnętrzne pochodzą z zaufanych repozytoriów.			✓
19.9	Upewnij się, że procesy budowania dla języków na poziomie systemu			✓

	mają włączone wszystkie flagi zabezpieczeń, takie jak ASLR, DEP i kontrole bezpieczeństwa.			
19.10	Upewnij się, że wszystkie komponenty aplikacji są hostowane przez aplikacje, tak jak biblioteki JavaScript, arkusze stylów CSS i czcionki web są hostowane przez aplikacje a niżeli przez CDN lub zewnętrznego dostawcę.			✓
19.11	Sprawdź, czy wszystkie składniki aplikacji, usługi i serwery używają własnego konta usługi o niskim poziomie uprawnień, które nie jest współużytkowane między aplikacjami ani nie jest używane przez administratorów.		✓	✓

20. Wymagania weryfikacji Internetu Rzeczy

Ta sekcja zawiera kontrole, które są specyficzne dla urządzenia Embedded / IoT. Te kontrole muszą być uwzględnione w połączeniu ze wszystkimi pozostałymi częściami odpowiedniego poziomu weryfikacji ASVS.

Cel:
Urządzenia Embedded/IoT powinny:

- mieć taki sam poziom kontroli bezpieczeństwa wewnątrz urządzenia, jaki mają na serwerze, poprzez wymuszanie kontroli bezpieczeństwa w zaufanym środowisku.
- Wrażliwe dane powinny być przechowywane na urządzeniu w bezpieczny sposób.
- Wszystkie wrażliwe dane przesyłane z urządzenia powinny wykorzystywać zabezpieczenia warstwy transportowej.

	Opis	1	2	3
20.1	Sprawdź, czy interfejsy debugowania warstwy aplikacji, takie jak USB lub serial, są wyłączone.	✓	✓	✓
20.2	Sprawdź, czy klucze kryptograficzne są unikalne dla każdego urządzenia	✓	✓	✓
20.3	Jeśli ma to zastosowanie, to sprawdź, czy elementy sterujące ochroną pamięci, takie jak ASLR i DEP, są włączone w systemie operacyjnym embedded / IoT.	✓	✓	✓
20.4	Sprawdź, czy interfejsy debugowania na chipie, takie jak JTAG lub SWD, są wyłączone lub czy dostępny mechanizm ochrony jest włączony i odpowiednio skonfigurowany.	✓	✓	✓

20.5	Sprawdź, czy w urządzeniu nie ma fizycznych nagłówków debugowania.	✓	✓	✓
20.6	Sprawdź, czy poufne dane nie są przechowywane niezaszyfrowane na urządzeniu.	✓	✓	✓
20.7	Sprawdź, czy urządzenie zapobiega wyciekom poufnych informacji.	✓	✓	✓
20.8	Sprawdź, czy aplikacje firmware chronią dane podczas przesyłu za pomocą zabezpieczeń transportowych.	✓	✓	✓
20.9	Sprawdź, czy aplikacje firmware sprawdzają podpis cyfrowy połączeń z serwerami.	✓	✓	✓
20.10	Sprawdź, czy komunikacja bezprzewodowa jest wzajemnie uwierzytelniana.	✓	✓	✓
20.11	Sprawdź, czy komunikacja bezprzewodowa jest wysyłana przez zaszyfrowany kanał.	✓	✓	✓
20.12	Sprawdź, czy aplikacje firmware przypinają podpis cyfrowy do zaufanego serwera(ów).		✓	✓
20.13	Sprawdzić obecność fizycznej odporności na ingerencję i / lub funkcji wykrywania sabotażu, w tym epoksydu (ang. epoxy).		✓	✓
20.14	Sprawdź, czy oznaczenia identyfikacyjne na chipach zostały usunięte.		✓	✓
20.15	Sprawdź, czy dostępne są wszystkie dostępne technologie ochrony własności intelektualnej dostarczone przez producenta chipów.		✓	✓
20.16	Sprawdź, czy istnieją mechanizmy kontroli bezpieczeństwa, które utrudniają odwrotną inżynierię oprogramowania firmware (np. usunięcie pełnych ciągów debugowania).		✓	✓
20.17	Sprawdź, czy urządzenie sprawdza poprawność podpisu boot image przed załadowaniem.		✓	✓
20.18	Sprawdź, czy proces aktualizacji firmware nie jest podatny na ataki time-of-check vs time-of-use.		✓	✓
20.19	Sprawdź, czy urządzenie używa podpisywania kodu i sprawdza poprawność plików aktualizacji oprogramowania przed instalacją.		✓	✓
20.20	Sprawdź, czy urządzenia nie można zmienić na starszą wersję prawidłowego firmware.		✓	✓
20.21	Sprawdź użycie kryptograficznie bezpiecznego generatora liczb		✓	✓

	pseudolosowych na urządzeniu wbudowanym (np. Za pomocą generatorów liczb losowych dostarczanych przez chipy).			
20.22	Sprawdź, czy urządzenie czyści firmware i dane poufne po wykryciu naruszenia lub odebraniu nieprawidłowego komunikatu.			✓
20.23	Sprawdź, czy używane są tylko mikrokontrolery obsługujące wyłączanie interfejsów debugowania (np. JTAG, SWD).			✓
20.24	Sprawdź, czy używane są tylko mikrokontrolery zapewniające istotną ochronę przed atakami typu "de-capping" i “side channel”.			✓
20.25	Sprawdź, czy poufne ślady użytkowania nie są wystawione na zewnętrzne warstwy obwodu drukowanego.			✓
20.26	Sprawdź, czy komunikacja między chipami jest szyfrowana.			✓
20.27	Sprawdź, czy urządzenie używa podpisywania kodu i waliduje kod przed wykonaniem.			✓
20.28	Sprawdź, czy poufne informacje przechowywane w pamięci są nadpisywane zerami, gdy tylko nie są już potrzebne.			✓
20.29	Sprawdź, czy aplikacje firmware wykorzystują “pojemniki” kernel do izolacji między aplikacjami.			✓