

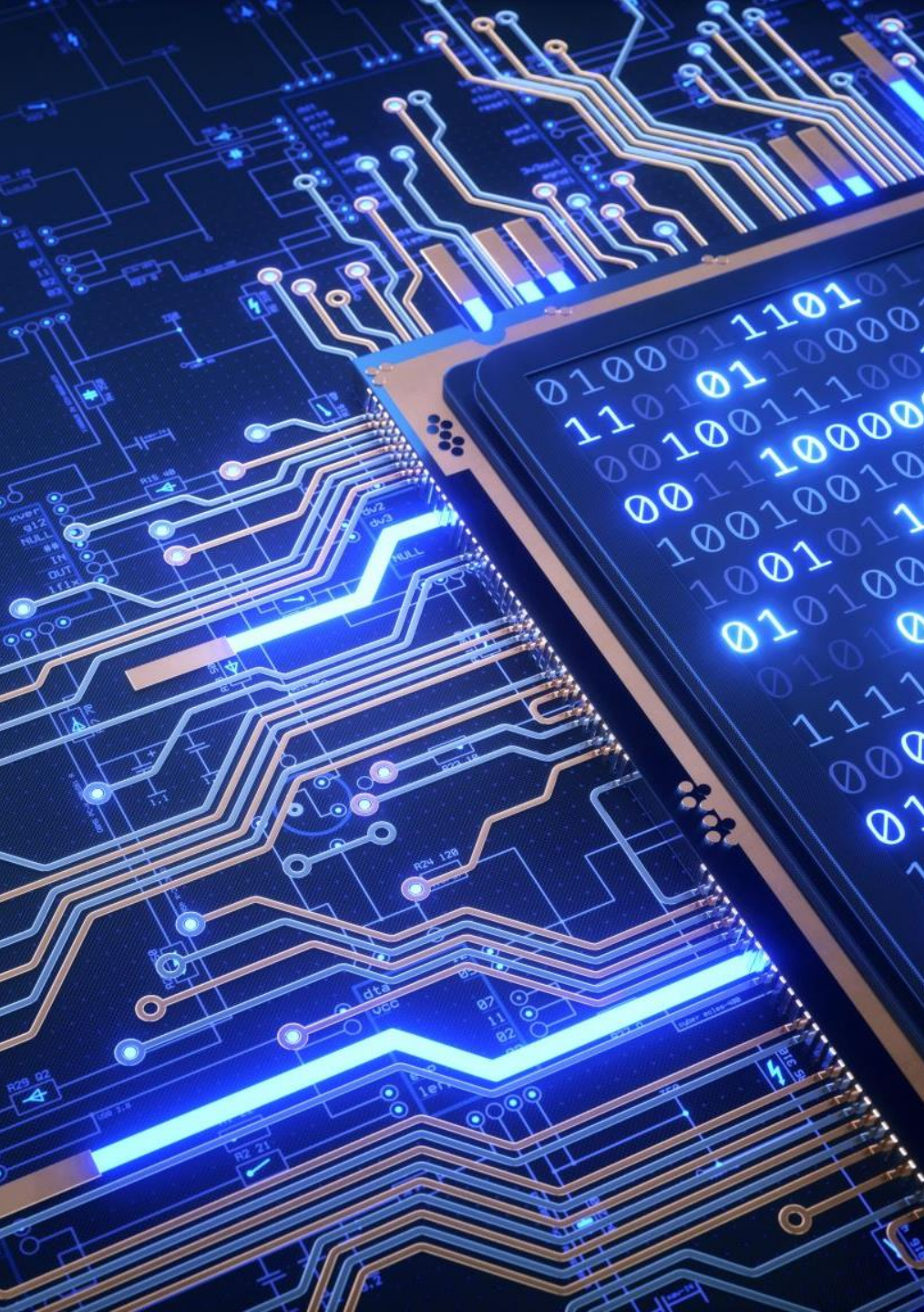
Bezpieczeństwo systemów i oprogramowania

Wykład 2

## Aspekty bezpieczeństwa w cyklu wytwarzania oprogramowania

autor: dr inż. Mariusz Sepczuk

e-mail: [mariusz.sepczuk@pw.edu.pl](mailto:mariusz.sepczuk@pw.edu.pl)



# Plan prezentacji

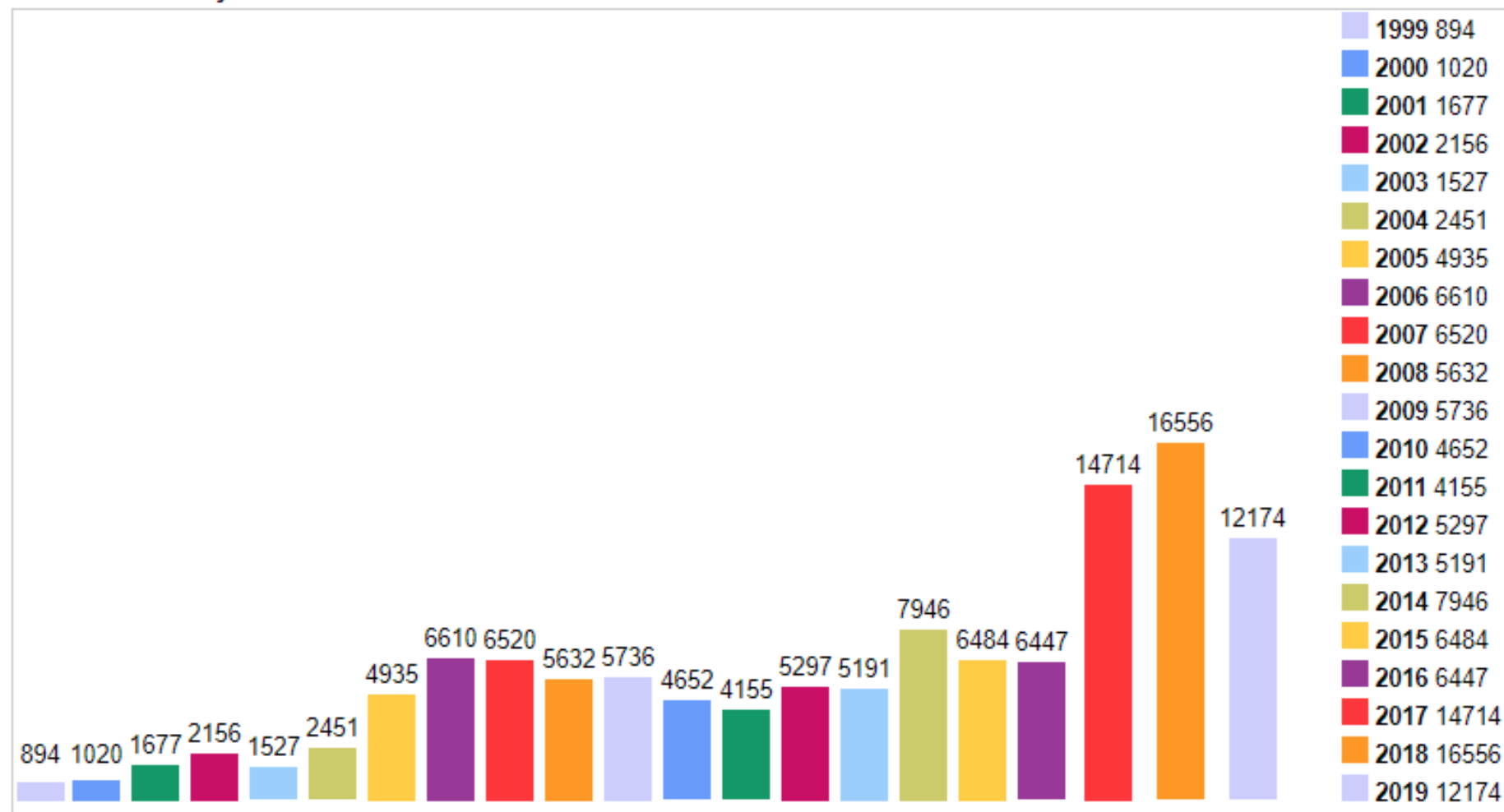
1. Typy problemów bezpieczeństwa związane z oprogramowaniem
2. Cykl wytwarzania oprogramowania
3. Cykl wytwarzania bezpiecznego oprogramowania
4. Modelowanie zagrożeń

# (Nie)bezpieczne oprogramowanie

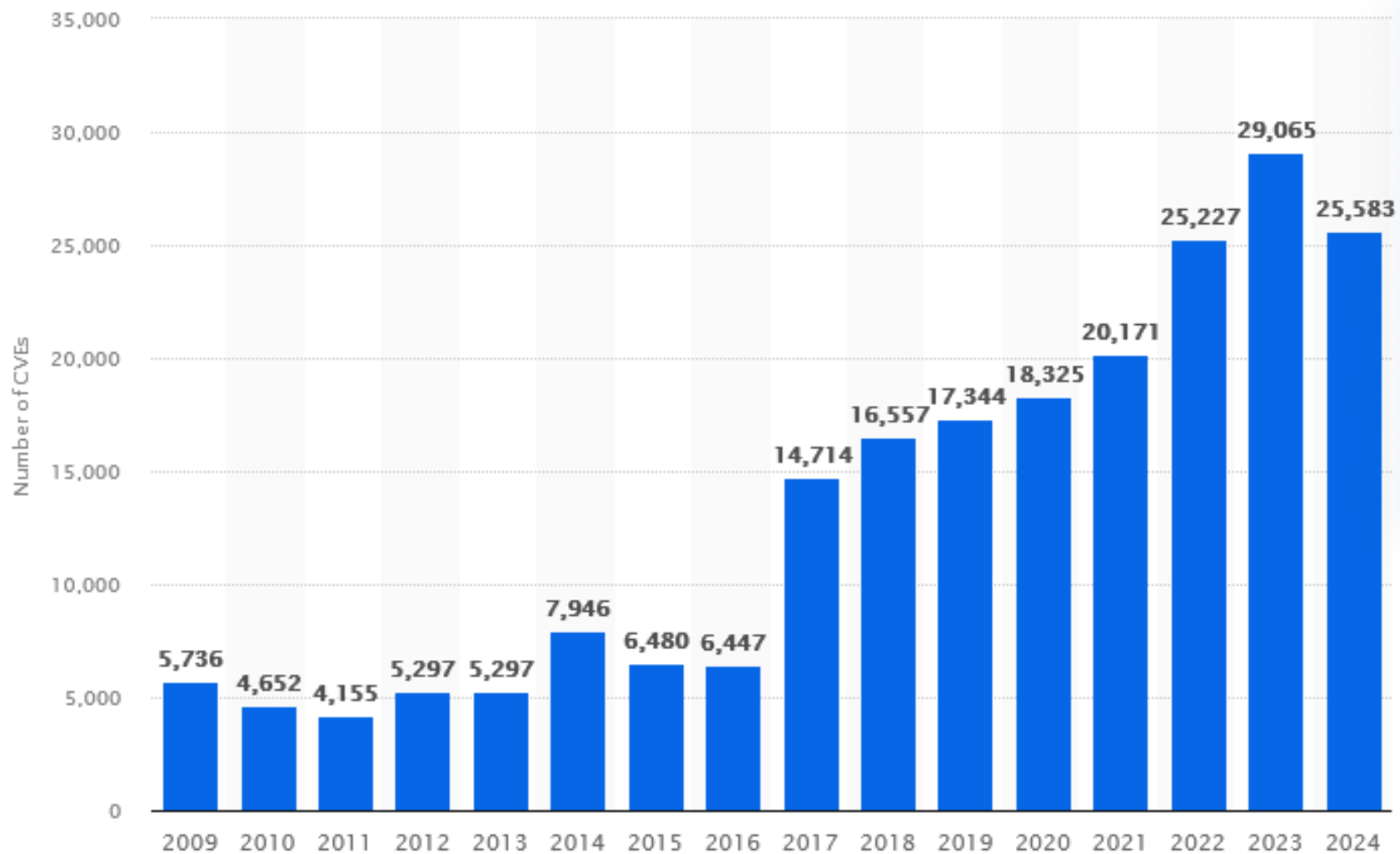
- Dlaczego strony internetowe, serwery, przeglądarki, laptopy, telefony komórkowe, punkty dostępu Wi-Fi, routery sieciowe, samochody, rozruszniki serca, sieć elektryczna, urządzenia do wzbogacania uranu, ... mogą zostać zhakowane?
- **Bo wszystkie zawierają oprogramowanie !!!**
- Oprogramowanie jest najsłabszym ogniwem w łańcuchu bezpieczeństwa, z możliwym wyjątkiem „czynnika ludzkiego”
- Hakerzy atakują:
  - **Oprogramowanie**
  - **Ludzi**
  - **Oprogramowanie + ludzi**
  - Crypto
  - Sprzęt
  - ...

# Skala problemów bezpieczeństwa (1/2)

Vulnerabilities By Year



# Skala problemów bezpieczeństwa (2/2)



# Zmiana charakteru atakującego

- Tradycyjnie hakerzy byli amatorami motywowanymi „zabawą”
  - Publikowanie ataków dla prestiżu
- W dzisiejszych czasach hakerzy są profesjonalni
  - Napastnicy schodzą do podziemia, bo zero-days są warte sporo pieniędzy
- Głównymi aktorami w dzisiejszych czasach są:
  - (zorganizowani) przestępcy z dużymi pieniędzmi i (wynajętą) wiedzą
    - Oprogramowanie ransomware i bitcoiny, które zmieniają zasady gry
  - Podmioty państwowe z jeszcze większymi pieniędzmi i wewnętrzną wiedzą fachową



# Ceny za zero-day na Desktopy/ Serwer

## ZERODIUM Payouts for Desktops/Servers\*

# ZERODIUM Payouts for Desktops/Servers\*

Windows

macOS

Linux/BSD

Any OS

RCE: Remote Code Execution

LPE: Local Privilege Escalation

SBX: Sandbox Escape or Bypass

VME: Virtual Machine Escape

Up to \$1,000,000											1.001 Win RCE Zero Click Win
Up to \$500,000								3.001 Chrome RCE+LPE Win	2.001 Apache RCE Linux	2.002 MS IIS RCE Win	
Up to \$250,000							5.001 MS Outlook RCE Win	4.001 MS Exchange RCE Win	2.003 OpenSSL RCE Linux	2.004 PHP RCE Linux	
Up to \$200,000	6.001 VMware ESXi VME Win/Linux	5.002 Thunderbird RCE Win/Linux				4.002 Sendmail RCE Linux	4.003 Postfix RCE Linux	4.004 Dovecot RCE Linux	4.005 Exim RCE Linux	2.005 nginx RCE Linux	
Up to \$100,000		3.002 Safari RCE+LPE Mac	3.003 Edge RCE+LPE Win	3.004 Firefox RCE+LPE Win	5.003 Word/Excel RCE Win	7.001 WordPress RCE Linux	7.002 cPanel/WHM RCE Linux	7.003 Plesk RCE Linux	7.004 Webmin RCE Linux		
Up to \$80,000	6.002 VMware WS VME Win/Linux					5.004 Adobe PDF RCE+SBX Win	5.005 WinRAR RCE Win	5.006 7-Zip RCE Win	6.003 Windows LPE/SBX Win		
Up to \$50,000	6.004 USB LPE Win/Mac	8.001 Antivirus RCE Win			5.007 WinZip RCE Win	5.008 tar RCE Linux	6.005 macOS LPE/SBX Mac	6.006 Linux LPE Linux	6.007 BSD LPE BSD		
Up to \$10,000	9.001 Routers RCE	8.002 Antivirus LPE Win	7.005 phpBB RCE Linux	7.006 vBulletin RCE Linux	7.007 MyBB RCE Linux	7.008 Joomla RCE Linux	7.009 Drupal RCE Linux	7.010 Roundcube RCE Linux	7.011 Horde RCE Linux		

\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/01 © zerodium.com

# Ceny za zero-day na telefony

## ZERODIUM Payouts for Mobiles\*

FCP: Full Chain with Persistence  
RCE: Remote Code Execution  
LPE: Local Privilege Escalation  
SBX: Sandbox Escape or Bypass

■ iOS  
■ Android  
■ Any OS

ZERODIUM Payouts for Mobiles\*

FCP: Full Chain with Persistence  
RCE: Remote Code Execution  
LPE: Local Privilege Escalation  
SBX: Sandbox Escape or Bypass

iOS  
 Android  
 Any OS

Up to \$2,500,000

Up to \$2,000,000

Up to \$1,500,000

Up to \$1,000,000

Up to \$500,000

Up to \$200,000

Up to \$100,000

1.001  
Android FCP  
Zero Click  
Android

1.002  
iOS FCP  
Zero Click  
iOS

2.001  
WhatsApp  
RCE+LPE  
Zero Click  
iOS/Android

2.002  
iMessage  
RCE+LPE  
Zero Click  
iOS

2.003  
WhatsApp  
RCE+LPE  
iOS/Android

2.004  
SMS/MMS  
RCE+LPE  
iOS/Android

3.001  
Persistence  
iOS

2.005  
WeChat  
RCE+LPE  
iOS/Android

2.006  
iMessage  
RCE+LPE  
iOS

2.007  
FB Messenger  
RCE+LPE  
iOS/Android

2.008  
Signal  
RCE+LPE  
iOS/Android

2.009  
Telegram  
RCE+LPE  
iOS/Android

2.010  
Email App  
RCE+LPE  
iOS/Android

4.001  
Chrome  
RCE+LPE  
Android

4.002  
Safari  
RCE+LPE  
iOS

5.001  
Baseband  
RCE+LPE  
iOS/Android

6.001  
LPE to  
Kernel/Root  
iOS/Android

2.011  
Media Files  
RCE+LPE  
iOS/Android

2.012  
Documents  
RCE+LPE  
iOS/Android

4.003  
SBX  
for Chrome  
Android

4.004  
Chrome RCE  
w/o SBX  
Android

4.005  
SBX  
for Safari  
iOS

4.006  
Safari RCE  
w/o SBX  
iOS

7.001  
Code Signing  
Bypass  
iOS/Android

5.002  
WiFi  
RCE  
iOS/Android

5.003  
RCE  
via MitM  
iOS/Android

6.002  
LPE to  
System  
Android

8.001  
Information  
Disclosure  
iOS/Android

8.002  
[k]ASLR  
Bypass  
iOS/Android

9.001  
PIN  
Bypass  
Android

9.002  
Passcode  
Bypass  
iOS

9.003  
Touch ID  
Bypass  
iOS

\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

2019/09 © zerodium.com



# Przykład 1: Uwierzytelnienie użytkownika

- Czy ten kod jest bezpieczny?

```
boolean verify (char[] input, char[] passwd , byte len) {  
    // No more than triesLeft attempts  
    if (triesLeft < 0) return false ; // no authentication  
    // Main comparison  
    for (short i=0; i <= len; i++)  
        if (input[i] != passwd[i]) {  
            triesLeft-- ;  
            return false ; // no authentication  
        }  
    // Comparison is successful  
    triesLeft = maxTries ;  
    return true ; // authentication is successful  
}
```

- Co lub przed czym chcemy chronić?

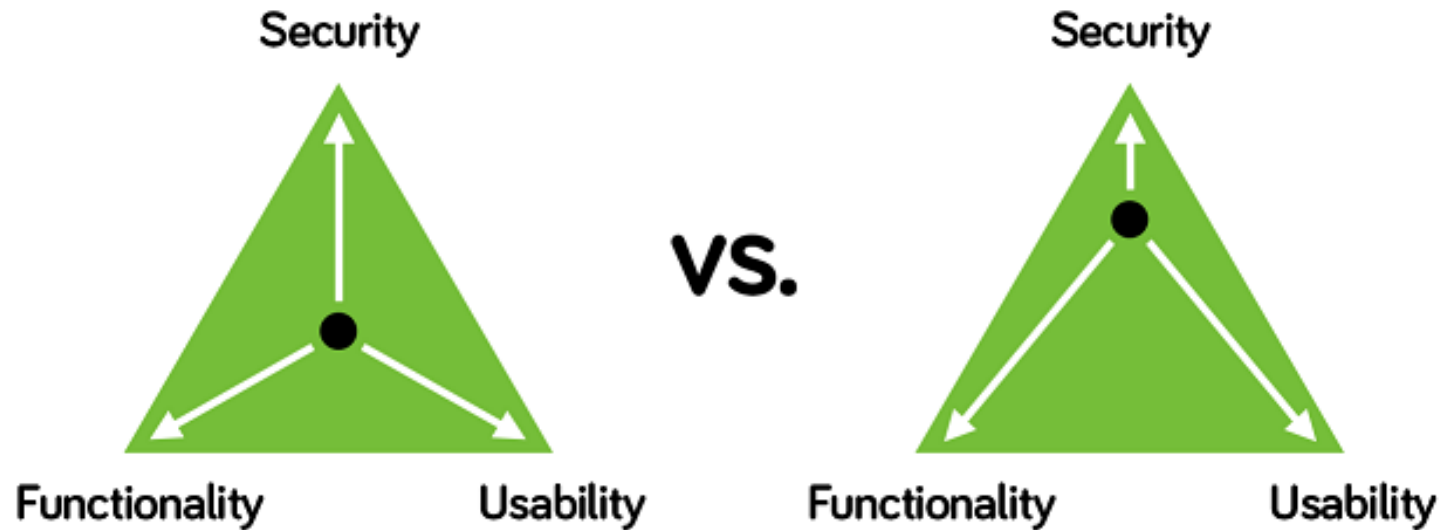
- Poufność hasła
- Wyciek danych
- Niespodziewane zachowanie programu
- Integralność kodu
- itp.

## (Nie)bezpieczne oprogramowanie: fakty

- Nie ma prostych środków na skomplikowane problemy
  - Kryptografia lub specjalne funkcje bezpieczeństwa nie rozwiązują magicznie wszystkich problemów
  - “if you think your problem can be solved by cryptography, you do not understand cryptography and you do not understand your problem” [Bruce Schneier]
- Bezpieczeństwo to nowa właściwość całego systemu - podobnie jak jakość
  - Często funkcjonalność przeważa nad bezpieczeństwem
- (Niefunkcjonalne) aspekty bezpieczeństwa powinny być integralną częścią projektu od samego początku

# Security, Functionality & Usability Triangle

- Poziom bezpieczeństwa w dowolnym systemie może być zdefiniowany za pomocą wpływu trzech komponentów:



# Typy problemów bezpieczeństwa w oprogramowaniu cz.1

Terminologia może być bardzo zagmatwana

Bug – błąd wprowadzony do oprogramowania:

- Na poziomie pisania kodu
- Na poziomie kompilacji

Flaw – błąd wprowadzony na etapie tworzenia koncepcji

- Nawet jeśli kod programu został napisany zgodnie z przyjętymi założeniami może posiadać błędy

Bug	Flaw
Buffer overflow: stack smashing Buffer overflow: one-stage attacks Buffer overflow: string format attacks Race conditions: TOCTOU Unsafe environment variables Unsafe system calls (fork(), exec(), system()) Incorrect input validation (black list vs. white list)	Method over-riding problems (subclass issues) Compartmentalization problems in design Privileged block protection failure (DoPrivilege()) Error-handling problems (fails open) Type safety confusion error Insecure audit log design Broken or illogical access control (role-based access control [RBAC] over tiers) Signing too much code

# Typy problemów bezpieczeństwa w oprogramowaniu cz. 2

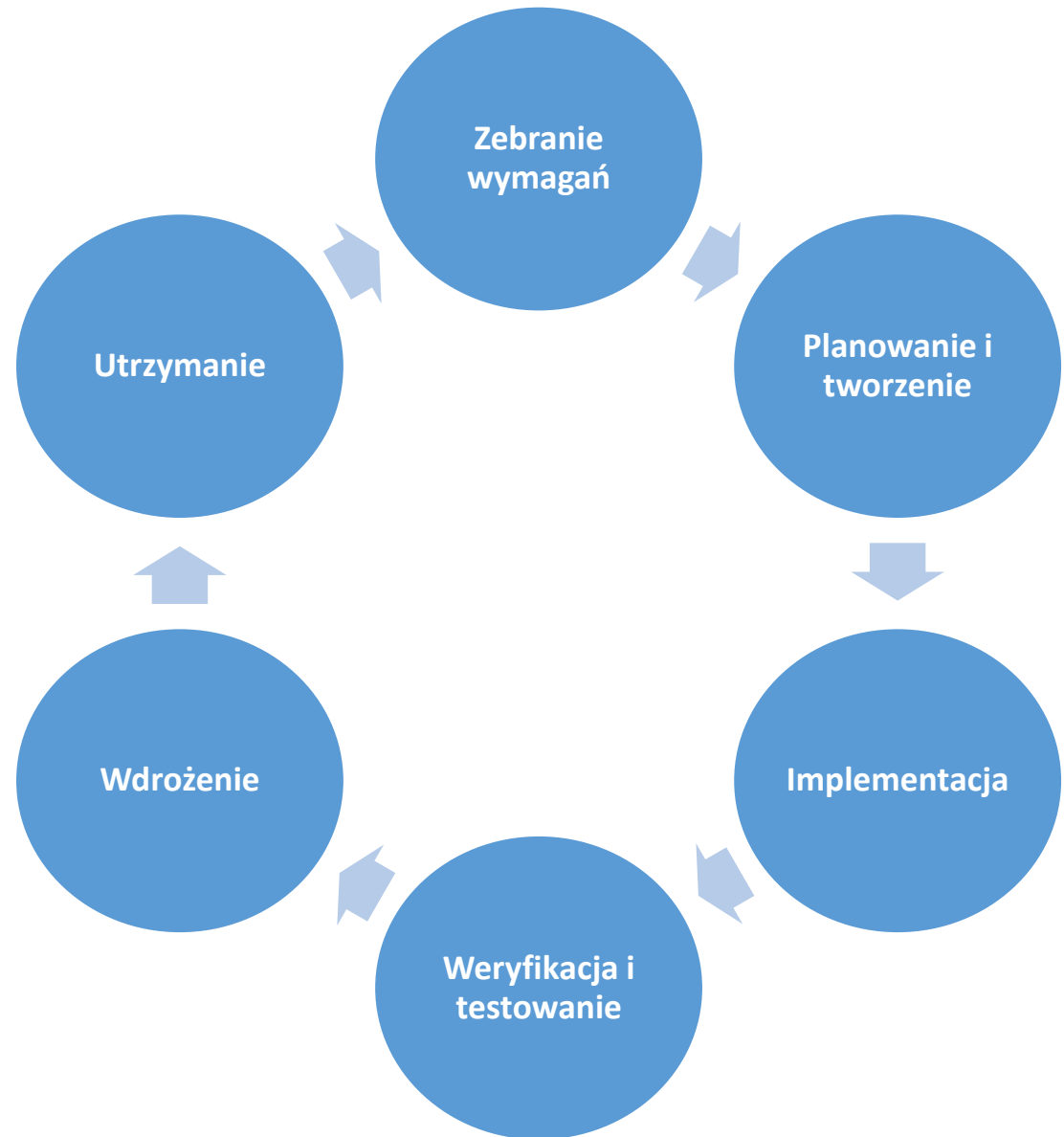
- **Podatność** – słabość, która otwiera drzwi do naruszeń bezpieczeństwa
  - Podatność nieeksploituwalna: atakujący nie może użyć błędu, aby skompromitować system
  - Podatność eksploituwalna: błąd może zostać użyty do opracowania ataku
- **Eksploit** – program umożliwiający wykorzystanie luki w zabezpieczeniach (z punktu widzenia atakującego!)
- **PoC eksploita** – atak na komputer lub sieć, który jest wykonywany tylko w celu udowodnienia, że można to zrobić
  - Zwykle nie powoduje żadnych szkód, ale pokazuje, w jaki sposób haker może wykorzystać lukę w oprogramowaniu lub sprzęcie
- **Złośliwe oprogramowanie** – fragment kodu „wstrzyknięty” do komputera w celu jego uszkodzenia
  - Zazwyczaj wykorzystują istniejące luki w zabezpieczeniach

# Jak tworzyć oprogramowanie

- Oprogramowanie jest wytwarzane zgodnie z pewnymi regułami
- Cykl Życia Rozwoju Oprogramowania (Software Development Life Cycle) to proces wytwarzania oprogramowania w kilku określonych kluczowych etapach, mający na celu wdrożenia jakości i wydajności
- Proces SDLC nie ma jednej ustalonej postaci
  - Najczęściej jest opisywany jako iteracyjne fazy
- Standardowo nie są w nim uwzględnione kwestie bezpieczeństwa
  - Dlatego wprowadzono S-SDLC (Secure Software Development Life Cycle)



# Etapy SDLC



# S-SDLC

- S-SDLC to zbiór najlepszych praktyk skoncentrowanych na dodaniu zabezpieczeń do standardowego SDLC
  - Uwzględnienie wymagań bezpieczeństwa już na etapie zbierania wymagań
- Problemy bezpieczeństwa mogą być rozwiązane na długo przed wdrożeniem aplikacji
  - Zmniejsza to ryzyko znalezienia luk w zabezpieczeniach aplikacji i minimalizuje wpływ ich znalezienia
- Celem S-SDLC nie jest całkowite wyeliminowanie tradycyjnych kontroli bezpieczeństwa, takich jak testy penetracyjne, ale raczej włączenie bezpieczeństwa w zakres obowiązków programistów i umożliwienie im tworzenia bezpiecznych aplikacji od samego początku

# Etapy S-SDLC

Zebranie wymagań	Planowanie i tworzenie	Implementacja	Weryfikacja i testowanie	Wdrożenie	Utrzymanie
<ul style="list-style-type: none"><li>• Wymagania bezpieczeństwa</li><li>• Wymagania dotyczące zgodności (compliance) np. PCI DSS, HIPPA, GDPR</li></ul>	<ul style="list-style-type: none"><li>• Modelowanie ryzyka</li><li>• Przegląd projektu i architektury rozwiązania pod kątem bezpieczeństwa</li></ul>	<ul style="list-style-type: none"><li>• Statyczna analiza bezpieczeństwa aplikacji (SAST)</li></ul>	<ul style="list-style-type: none"><li>• Dynamiczna analiza bezpieczeństwa aplikacji (DAST)</li><li>• Testy penetracyjne</li></ul>	<ul style="list-style-type: none"><li>• Utwardzanie (hardening) środowiska</li><li>• Testy penetracyjne</li><li>• Procedura rollback'u</li></ul>	<ul style="list-style-type: none"><li>• Cykliczna ocena bezpieczeństwa</li><li>• Obsługa incydentów</li><li>• Monitorowanie aplikacji</li></ul>

# Krok 0: Szkolenie w zakresie bezpieczeństwa

- Szkolenie w zakresie zwiększenia świadomości bezpieczeństwa dla programistów (i nie tylko) powinno obejmować:
  - Najlepsze praktyki bezpiecznego programowania oparte na OWASP Top 10
  - Techniki obronne(np. OWASP ESAPI)
  - Zagrożenia związane z Web Service i Web 2.0
  - Narzędzia do oceny bezpieczeństwa aplikacji typu „black box”
  - Najlepsze praktyki dotyczące przeglądów kodu pod kątem bezpieczeństwa
  - Kryptografia, haszowanie i zaciemnianie
  - Strategie walidacji danych wejściowych
  - Zrozumienie zagrożeń: modelowanie zagrożeń

# Krok 1: Zbieranie wymagań

- Wymagania dotyczące nowych funkcji są zbierane od różnych interesariuszy
- Ważne jest, aby zidentyfikować wszelkie kwestie bezpieczeństwa dotyczące wymagań funkcjonalnych, które są zbierane dla nowego oprogramowania
- **Przykładowe wymaganie funkcjonalne:** użytkownik potrzebuje możliwość zweryfikowania swoich danych kontaktowych, zanim będzie mógł odnowić członkostwo
- **Przykładowe wymaganie bezpieczeństwa:** użytkownicy powinni mieć wgląd tylko do swoich danych kontaktowych i nikogo innego

## Krok 2: Planowanie i tworzenie

- Przełożenie wymagań objętych zakresem na plan określający, jak powinny one wyglądać w rzeczywistej aplikacji
  - Np. wybór technologii, frameworku i języka
- Tutaj też dokonywana jest analiza rozwiązania pod kątem bezpieczeństwa
- Wymagania funkcjonalne zazwyczaj opisują, co powinno się wydarzyć, podczas gdy wymagania dotyczące bezpieczeństwa zwykle koncentrują się na tym, co nie powinno
- **Przykładowe wymaganie funkcjonalne:** strona powinna pobierać nazwę użytkownika, e-mail, telefon i adres z tabeli CUSTOMER\_INFO w bazie danych i wyświetlać je na ekranie
- **Przykładowe wymaganie bezpieczeństwa:** przed pobraniem informacji z bazy danych musimy sprawdzić, czy użytkownik ma ważny token sesji. W przypadku braku, użytkownik powinien zostać przekierowany na stronę logowania



# Podstawowe pojęcia związane z ryzykiem

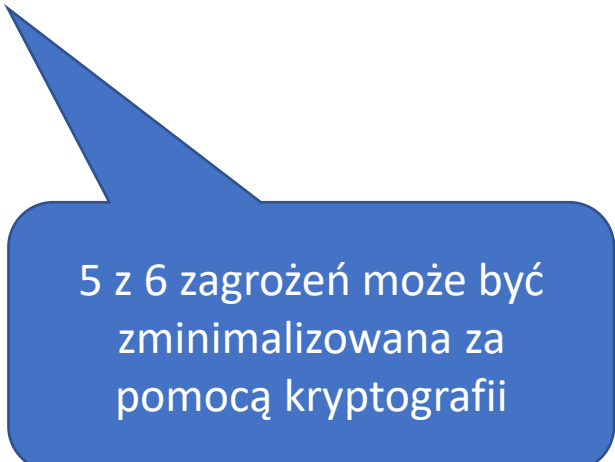
- **Zagrożenie (ang. Threat)** – każde potencjalne wystąpienie, które może mieć niepożądany wpływ na zasoby systemowe
  - Niepożądanymi efektami mogą być awaria systemu, możliwość odczytu poufnego pliku lub modyfikacji klucza rejestru itp.
- **Podatność (ang. Vulnerability)** – cecha, która umożliwia wystąpienie zagrożenia
  - Typowymi przykładami są przepełnienie bufora, odblokowany port na firewall, możliwość wstrzyknięcia kodu SQL w polu do wyszukiwania na stronie www
- **Atak (ang. Attack)** – działanie podejmowane przez złośliwego intruza, który wykorzystuje podatności w celu materializacji zagrożenia
  - Typowe ataki to usunięcie struktury bazy danych po wstrzyknięciu kodu w okno wyszukiwania lub ciągły reset aplikacji wynikający z przepełnienia bufora

# Modelowanie zagrożeń

- Modelowanie zagrożeń (ang. Threat Modeling) swoim zakresem obejmuje:
  - Identyfikację zasobów, które mogą być atakowane
  - Analizę zagrożeń i podatności atakowanego zasobu
  - Ustalenie ryzyka
  - Propozycja środków naprawczych

# Analiza zagrożeń

- Klasyfikacja zagrożeń **STRIDE**:
  - Przejęcie i podszycie się pod czyjąś tożsamość (**S**poofing Identity)
  - Manipulowanie danymi (**T**ampering with Data)
  - Zaprzeczanie autorstwu operacji (**R**epudiation)
  - Ujawnienie informacji (**I**nformation Disclosure)
  - Ataki odmowy usługi (**D**enial of Service)
  - Podniesienie poziomu uprawnień (**E**levation of Privilege)



5 z 6 zagrożeń może być zminimalizowana za pomocą kryptografii

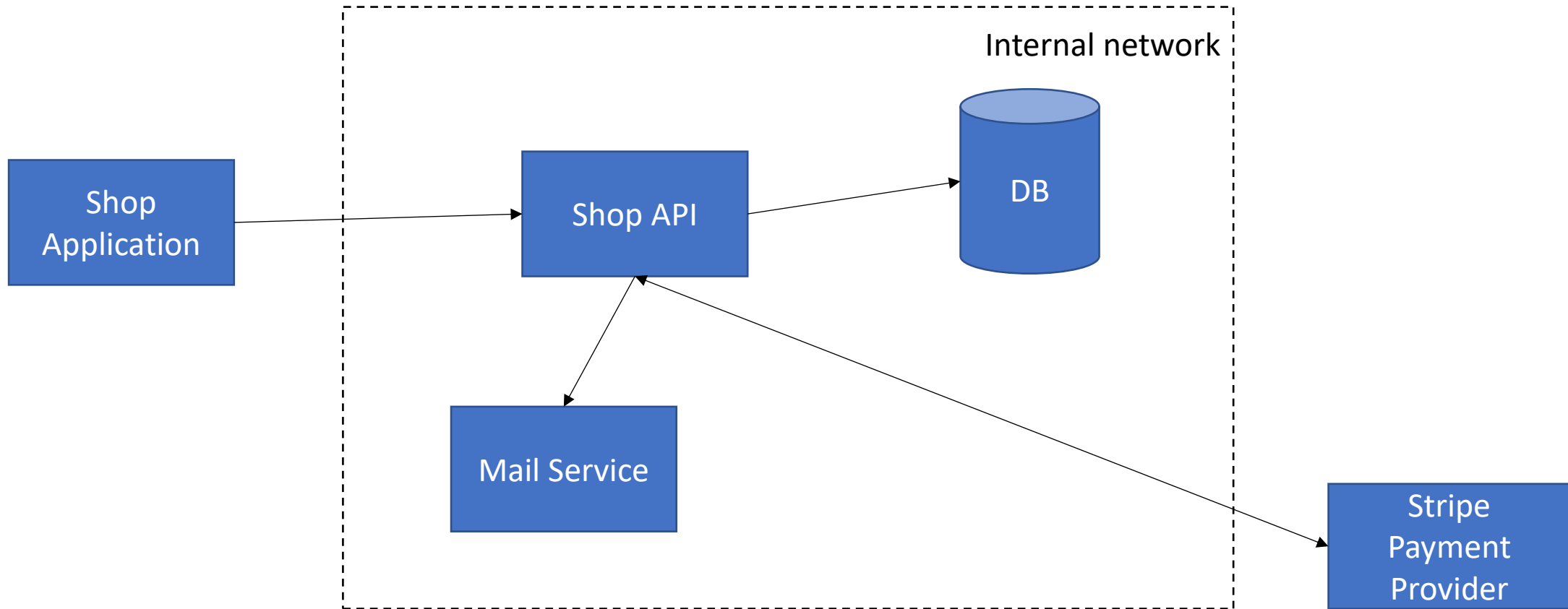
# Zrozumienie zagrożeń

Zagrożenie	Właściwość	Definicja	Przykład
Podszywanie się	Uwierzytelnienie	Podszywanie się pod kogoś lub coś	Udawanie że jest się dyrektorem albo witryną google.com, albo bibliotekę ntdll.dll
Modyfikacja	Integralność	Modyfikacja danych lub kodu	Modyfikacja pliku DLL na dysku lub innym nośniku pamięci, modyfikacja pakietu w trakcie przesyłania przez sieć LAN
Zaprzeczenie, wyparcie się	Niezaprzeczalność	Utrzymywanie, że nie wykonało się określonych akcji	„Ale to nie ja wysłałem tego maila”, „Nie zmodyfikowałem pliku”, „Przecież ja nie oglądam takich stron internetowych”
Ujawnienie informacji	Poufność	Ekspozycja informacji w sposób umożliwiający ich odczytanie przez osoby do tego nieuprawnione	Opublikowanie listy klientów w witrynie internetowej, umożliwienie odczytu tajnych plików
Odmowa usługi	Dostępność	Uniemożliwienie dostępu do usługi uprawnionym użytkownikom	Destabilizacja pracy systemu lub usługi sieciowej, rozsyłanie pakietów i zużywanie cennych sekund procesora, lub przekierowywanie pakietów w niebyt
Podniesienie poziomu uprawnień	Autoryzacja	Uzyskanie możliwości działania bez właściwej autoryzacji	Uzyskanie uprawnień administratora przy korzystaniu z konta zwykłego użytkownika, zdalne wykonanie kodu przez nieuwierzytelnionych użytkowników

# Modelowanie zagrożeń w praktyce

- Założmy, że nasz zespół buduje klasyczny sklep internetowy
- Posiada on proste funkcje w stylu:
  - przeglądanie artykułów,
  - dodawanie do koszyka,
  - zamawianie,
  - obsługa płatności online,
  - wysyłka potwierdzeń mailowych.

# Diagram sklepu internetowego





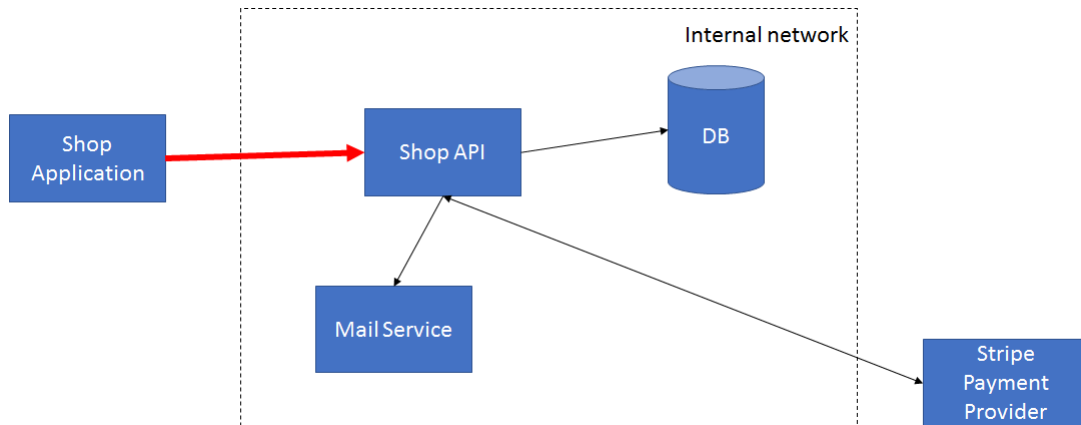
# Co analizować i jak analizować

- Nie dla każdego elementu musimy analizować zagrożenia

Element diagramu/systemu	S	T	R	I	D	E
Proces/wewnętrzny komponent	Tak	Tak	Tak	Tak	Tak	Tak
Zewnętrzny komponent	Tak		Tak			
Przesył danych		Tak		Tak	Tak	
Magazyn danych		Tak	Tak	Tak	Tak	

# Analiza krok po kroku

- Rozważamy przepływ danych pomiędzy aplikacją frontendową, a API



- Tampering
  - ktoś może zmodyfikować cenę artykułu podczas wysyłki podsumowania danych do serwera,
  - ktoś może umieścić wektor ataku SQL Injection w polach imię i nazwisko.
- Information Disclosure
  - ktoś może zmodyfikować id w treści zapytania i w ten sposób zdobyć dostęp do nie swoich danych,
  - ktoś może wpiąć się atakiem Man in the Middle w niezabezpieczone połączenie HTTP,
  - ktoś może wysłać zapytania zapisujące dane zasobu, do którego nie ma uprawnień.
- Denial of Service
  - ktoś może wysłać wiele takich samych zapytań, obciążając serwer

# Ustalenie ryzyka

- Ocena ryzyka może być przeprowadzona za pomocą metody DREAD
- DREAD składa się z pięciu składowych:
  - Damage – Obrażenia – jak zły byłby atak?
  - Reproducibility – Odtwarzalność – jak łatwo odtworzyć atak?
  - Exploitability – Możliwość wykorzystania – ile pracy wymaga uruchomienie ataku?
  - Affected users – Dotknięci użytkownicy – na ile osób to wpłynie?
  - Discoverability – Wykrywalność – jak łatwo jest wykryć zagrożenie?
- Dla każdej składowej przyznajemy punkty ze zdefiniowanego przedziału np. [0-3]
- Miara oceny ryzyka jest średnią z ocen składowych

## Przykład 2

- Proszę ocenić następującą podatność przy wykorzystaniu metodyki oceny ryzyka DREAD (skala 0-3)
- **Scenariusz z podatnością:** Uwierzytelniony użytkownik w aplikacji może załadować plik z treścią multimedialną. Nazwa pliku wyświetlana jest w aplikacji. Jeżeli przechwyci się żądanie HTTP i zmieni nazwę pliku na zawierającą odpowiednio sformatowany fragment kodu JavaScript, istnieje możliwość kradzieży tokena sesji

# Przykładowa odpowiedź

- Poszczególne oceny mogą się nieco różnić
- Propozycja ocen DREAD:
  - Damage: **1** (można przejąć tożsamość użytkownika, ale tylko w aplikacji – zakładamy, że nie umożliwi to wykonanie kodu)
  - Reproducibility: **3** (można wywołać za każdym powtórzeniem)
  - Exploitability: **2** (eksploatacja jest możliwa, jeśli uwierzytelniony użytkownik odwiedzi konkretną podstronę z treścią, co jest prawdopodobne, ale nie musi zdarzyć się za każdym razem)
  - Affected users: **2** (dotyka wszystkich uwierzytelnionych użytkowników, którzy odwiedzą podatną podstronę)
  - Discoverability: **1** (podatność może zauważyć tylko uwierzytelniony użytkownik o złośliwych intencjach- konieczne użycie HTTP proxy)
- Wartość ryzyka wynosi **1,8** (ryzyko raczej **średnie**)

# Środki naprawcze

- Szukając środków naprawczych warto wziąć pod uwagę następujące rozwiązania:
  1. Czy możemy rozwiązać ten problem narzędziem już wykorzystywanym u nas w firmie? Być może problem ten został już rozwiązany w innych aplikacjach.
  2. Jakie rozwiązania danego problemu oferują standardy branżowe?
  3. Jak podobny problem został rozwiązany przez innych? Tu możemy szukać rozwiązań na blogach lub forach internetowych.



## Krok 3: Implementacja

- Pisanie oprogramowanie zgodnie z przyjętymi regułami dotyczącymi bezpiecznego kodowania
  - Programowanie defensywne
- Analiza kodu źródłowego przed kompilacją zapewnia skalowalną metodę przeglądu kodu bezpieczeństwa i pomaga zapewnić przestrzeganie zasad bezpiecznego kodowania
  - Manualna lub automatyczna (SAST)
- Reguły bezpiecznego pisania kodu mogą uwzględniać:
  - Używanie sparametryzowanych zapytań SQL i walidacja danych wejściowych
  - Upewnienie się, że programiści przeprowadzają odpowiednią walidację, aby zapobiec XSS
  - Sanityzacja danych pochodzących od użytkownika
  - Bezpieczne przechowywanie danych
  - Bezpieczna kontrola dostępu

# SAST

- Analiza statyczna jest wykonywana wyłącznie na kodzie źródłowym aplikacji bez jej wykonywania
  - Analiza może być wykonywana już na etapie pisania kodu!
- Dostępny kod źródłowy jest skanowany, a wszystkie problemy są wskazywane w dokładnym wierszu kodu w celu szybkiej naprawy
- Typy analizy statycznej:
  - **Analiza przepływu danych** (ang. „*Data Flow Analysis*”) — polega na sprawdzeniu zasięgu, czasu życia zmiennych oraz zależności między nimi.
  - **Taint Analysis** — polega na identyfikacji zmiennych kontrolowanych przez użytkownika, oznaczeniu ich (jako „*brudne*”), a następnie sprawdzeniu przez jakie funkcje zmienne te są wykorzystywane.
  - **Analiza leksykograficzna** (ang. „*Lexical Analysis*”) — jest to analiza gramatyczna kodu pod kątem wykrywania popularnych konstrukcji powodujących błędy bezpieczeństwa.

## Krok 4: Weryfikacja i testowanie

- Faza weryfikacji to etap, w którym aplikacje przechodzą dokładny cykl testowy, aby upewnić się, że spełniają pierwotny projekt i wymagania
  - Jest to również doskonałe miejsce do wprowadzenia zautomatyzowanych testów bezpieczeństwa przy użyciu różnych technologii (DAST)
- Testowanie może dotyczyć:
  - Vulnerability scanning/assessment
  - Testów bezpieczeństwa/penetracyjnych
  - Fuzzingu (rzadziej)

# DAST

- Analiza dynamiczna wykonuje skanowanie po stronie klienta danej aplikacji internetowej, która jest wdrożona i uruchomiona
- Złośliwe wzorce wejściowe dla typowych ataków www są automatycznie wysyłanych na adres URL aplikacji, podczas gdy jej odpowiedzi są oceniane pod kątem nietypowego zachowania, które może wskazywać na lukę
- Nie wszystkie podatności mogą zostać wykryte
  - W szczególności te które wymagają specyficznej kombinacji działań
- DAST jest często używana jako pomoc w ręcznych testach penetracyjnych.

# SAST vs DAST

SAST	DAST
Testy „white box” Tester ma dostęp do informacji o użytych technikach do budowy aplikacji. Aplikacja jest testowana wewnętrznie. Jest to podejście programisty.	Testy „black box” Tester nie ma dostępu do informacji o użytych technikach do budowy aplikacji. Aplikacja jest testowana z zewnątrz. Jest to podejście hackera.
Wymagany kod aplikacji SAST nie wymaga działającej aplikacji. Analizuje jej kod źródłowy lub binarny bez uruchamiania aplikacji	Wymagana działająca aplikacja DAST nie wymaga kodu źródłowego aplikacji. Analizują ją podczas jej działania.
Znajdowanie podatności na wczesnym etapie SDLC Skanowanie może zostać wykonane jak tylko napisany zostanie kawałek kodu realizujący jakąś funkcję.	Znajdowanie podatności w późnym etapie SDLC Podatności mogą zostać wykryte po ukończeniu cyklu programowania i uruchomieniu aplikacji.
Niskie koszty załatania podatności Luki w zabezpieczeniach znajdują się wcześniej w SDLC, a zatem łatwiej i szybciej można je naprawić.	Wysokie koszty załatania podatności Luki w zabezpieczeniach są wykrywane pod koniec SDLC, więc środki zaradcze często są przenoszone do następnego cyklu.
Nie można wykryć problemów związanych z działaniem aplikacji i środowiskiem Narzędzie skanuje kod statyczny, więc nie może wykryć luk w czasie wykonywania.	Można wykryć problemów związanych z działaniem aplikacji i środowiskiem Narzędzie wykorzystuje dynamiczną analizę aplikacji, jest w stanie znaleźć luki w zabezpieczeniach w czasie działania aplikacji.
Zwykle obsługuje wszystkie rodzaje oprogramowania Przykłady: aplikacje www, usługi sieciowe i grube klienty	Zwykle skanuje tylko aplikacje takie jak aplikacje www i usługi sieciowe DAST nie jest przydatny w przypadku innych rodzajów oprogramowania.

# Fuzzing

- Fuzzing to technika Black Box testowania oprogramowania, która polega na znajdowaniu błędów implementacji przy użyciu nieprawidłowo/częściowo zniekształconych wstrzyknięć danych w sposób zautomatyzowany
- Cel fuzzingu polega na założeniu, że w każdym programie znajdują się błędy, które czekają na wykrycie
  - Dlatego systematyczne podejście powinno je wcześniej czy później znaleźć
- Fuzzing to proces wysyłania celowo nieprawidłowych losowych danych do produktu w celu znalezienia przypadków testowych, które powodują awarię
  - Te błędy mogą prowadzić do możliwych do wykorzystania luk w zabezpieczeniach

## Krok 5: Wdrożenie

- Gdy aplikacja jest już „gotowa” do upublicznienia jest wdrażana
- Jeszcze przed wdrożeniem aplikacji musi zostać opracowane plan rollback'u
  - Wycofanie wdrożonych zmian
- Oczywiście po wdrożeniu aplikacji można na niej przeprowadzić testy penetracyjne
  - Jest to jednak obarczone ryzykiem „położenia” aplikacji
- Dodatkowo, należy się upewnić co do poprawności:
  - Konfiguracji serwera na którym znajduje się aplikacja
  - Konfiguracji sieci dzięki której będzie się można z aplikacją komunikować

## Krok 6: Utrzymanie

- Produkty muszą być stale aktualizowane, aby były zabezpieczone przed nowymi lukami
  - Testy bezpieczeństwa nie kończą się na etapie weryfikacji i testowania
- Podatności mogą znajdować się w kodzie napisanym przez programistów, ale coraz częściej można je znaleźć w podstawowych komponentach open source, które składają się na aplikację
  - Prowadzi to do wzrostu liczby podatności typu „zero-day”
- Luki w zabezpieczeniach mogą również pochodzić z innych źródeł, takich jak zewnętrzne testy penetracyjne przeprowadzane przez etycznych hakerów lub zgłoszenia od opinii publicznej za pośrednictwem tak zwanych programów „bug bounty”
- Dlatego aplikacja po wdrożeniu musi być nieustannie monitorowana, a zgłaszane incydenty szybko podjęte i naprawione



# Dobre praktyki S- SDLC

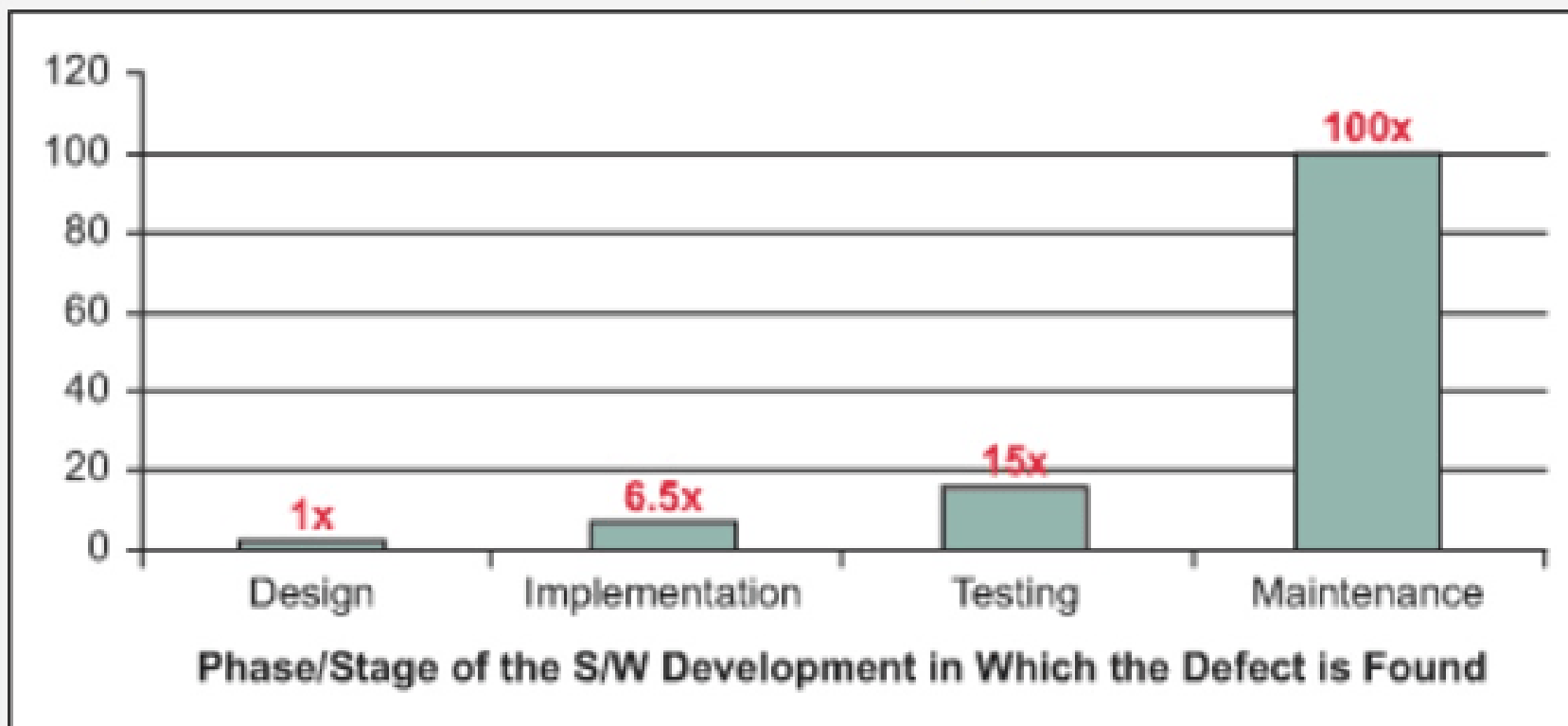
- Zdefiniuj politykę szkoleń w zakresie bezpieczeństwa, która odnosi się do wymagań w oparciu o obowiązki zawodowe
- Zdefiniuj wytyczne dotyczące wymagań bezpieczeństwa dla aplikacji
- Zdefiniuj i zastosuj praktyki oceny ryzyka
- Zdefiniuj wytyczne dotyczące projektowania bezpiecznej architektury
- Wykonaj modelowanie zagrożeń w fazie projektowania
- Zdefiniuj wytyczne dotyczące bezpiecznego kodowania
- Przeprowadź testy bezpieczeństwa aplikacji
- Zdefiniuj wytyczne i procedury bezpiecznego wdrażania
- Zdefiniuj wzorce bezpieczeństwa dla konfiguracji środowiska aplikacji

## Korzyści wynikające z używania S- SDLC

- Tworzenie bezpieczniejszego oprogramowanie
- Redukcja kosztów
- Wzrost świadomości interesariuszy projektu w kwestiach bezpieczeństwa
- Wczesna identyfikacja i naprawa luk w zabezpieczeniach
- Zmniejszanie ryzyka dla organizacji
- Szybkość

# Koszt naprawy błędu w zależności od etapu

Figure 1: Relative Costs to Fix Software Defects (Source: IBM Systems Sciences Institute)



# Metodyki dla S-SDLC

- Microsoft SDL
- OpenSAMM (Software Assurance Maturity Model)
- OWASP CLASP
- OWASP Touchpoints
- ...

Bezpieczeństwo systemów i oprogramowania

Wykład 2

## Aspekty bezpieczeństwa w cyklu wytwarzania oprogramowania

autor: dr inż. Mariusz Sepczuk

e-mail: [mariusz.sepczuk@pw.edu.pl](mailto:mariusz.sepczuk@pw.edu.pl)