

## Bezpieczeństwo infrastruktury krytycznej i systemów sterowania przemysłowego IoT - Laboratorium 9 – Threat Modeling

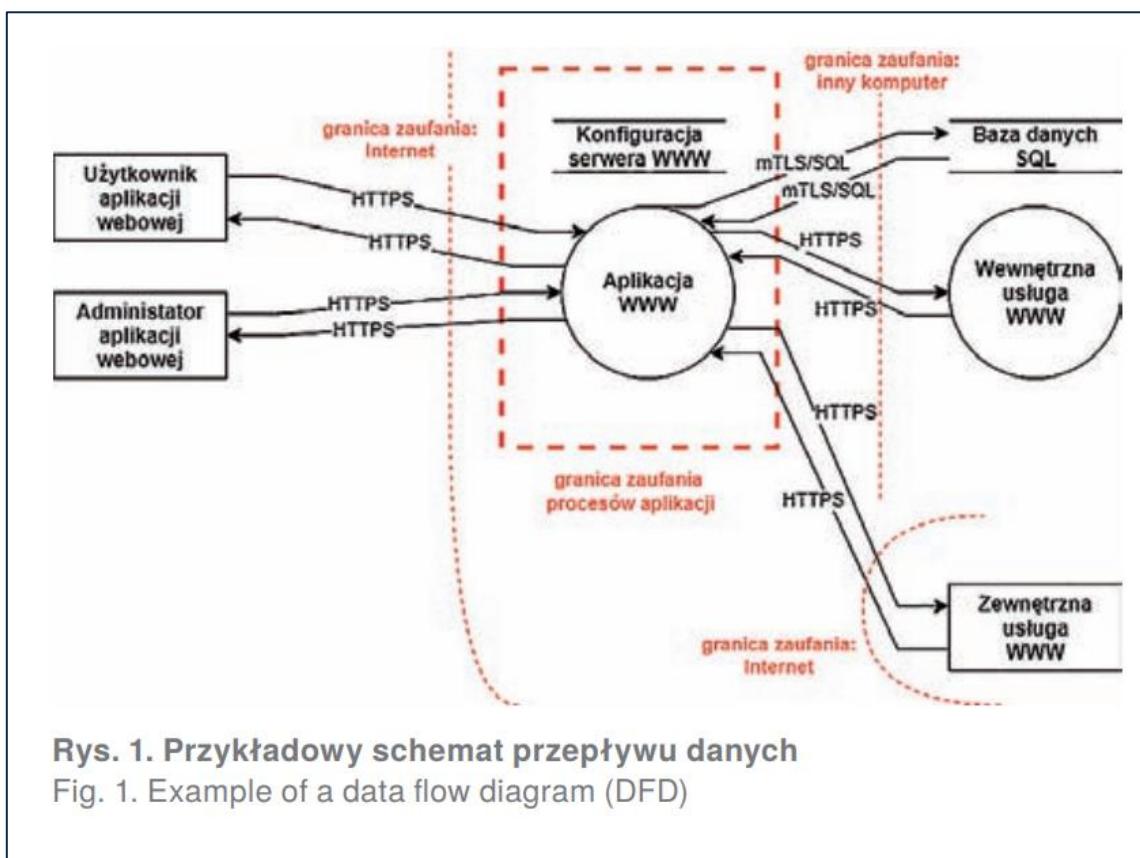
Forma: praca w samodzielna / małe zespoły (2 - 4 osoby)

### Wstęp teoretyczny

#### Diagram przepływu danych

- <https://kanbantool.com/pl/przewodnik-kanban/flowchart-diagram-przeplywu>
- <https://www.lucidchart.com/pages/pl/diagram-przeplywu-danych>
- <https://bibliotekanauki.pl/articles/2068668.pdf>

Metody modelowania zagrożeń rozwinięły wizualną reprezentację aplikacji i infrastruktury wykorzystującą diagramy przepływu danych DFD (ang. data flow diagram). Diagramy DFD zostały opracowane w latach 70. XX wieku jako narzędzie dla inżynierów systemowych do wysokopoziomowej reprezentacji sposobu w jaki aplikacja zarządza danymi: jak realizuje przepływy, przechowywanie oraz manipulowanie danymi w infrastrukturze, na której się wykonuje. Tradycyjnie, diagramy DFD wykorzystywały zaledwie cztery symbole: przepływy danych, magazyny danych, procesy i aktorów. Potem dodano dodatkowy symbol, tak zwane granice zaufania, by umożliwić wykorzystanie diagramów DFD właśnie do modelowania zagrożeń teleinformatycznych. Celowo ograniczony graficzny język opisu oraz ograniczona liczba dostępnych typów elementów sprzyja utrzymaniu dyscypliny opisu, a także wspiera uniwersalność i komunikatywność diagramów – następnie często badanych przez analityków bezpieczeństwa, którzy nie byli przecież ich autorami. Poprzez zastosowanie stosunkowo prostej reprezentacji graficznej unika się ryzyka wystąpienia przerostu formy prezentacji diagramu nad jego treścią. Po rozpisaniu systemu na pięć typów elementów, eksperci do spraw bezpieczeństwa analizują każdy zidentyfikowany punkt wejścia zagrożenia pod kątem wszystkich znanych kategorii zagrożeń. Po zidentyfikowaniu potencjalnych zagrożeń można ustalić środki zaradcze ograniczające zagrożenia lub przeprowadzić dodatkową analizę. Podsumowując, diagram przepływu danych może zawierać następujące typy elementów: – procesy – przepływy danych – interaktorzy – magazyny danych – granice zaufania

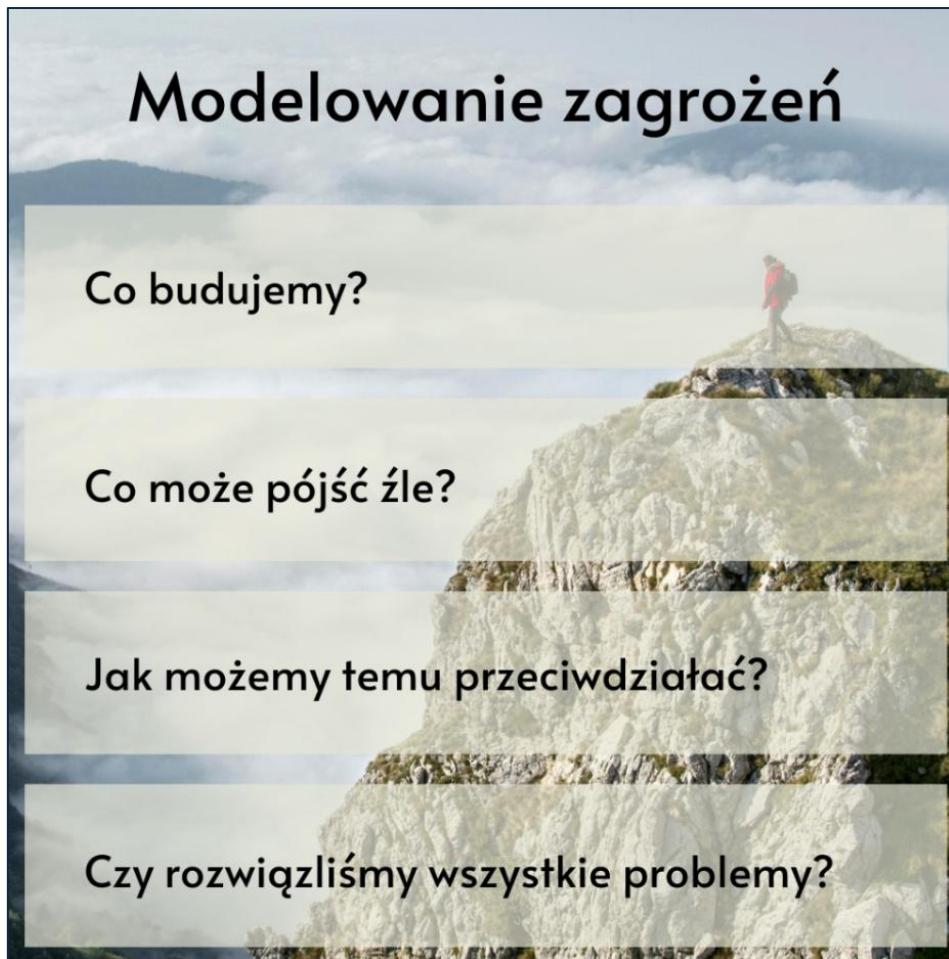


## **Modelowanie zagrożeń (Threat modeling)**

- <https://bibliotekanauki.pl/articles/2068668.pdf>
- [https://owasp.org/www-community/Threat\\_Modeling](https://owasp.org/www-community/Threat_Modeling)
- [https://owasp.org/www-community/Threat\\_Modeling\\_Process](https://owasp.org/www-community/Threat_Modeling_Process)
- [https://cheatsheetseries.owasp.org/cheatsheets/Threat\\_Modeling\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html)

U podstaw modelowania zagrożeń w systemach teleinformatycznych leżą cztery proste pytania:

1. Jaki jest przedmiot projektu?
2. Co może pójść nie tak?
3. Co zamierzamy z tym zrobić?
4. Czy właściwie wykonano ewaluację?



O potencjalnych intruzach, którzy mogą wykorzystać podatności wiadomo mniej niż o samym projekcie chronionego rozwiązania, stąd próba trafnego opisania możliwych zachowań potencjalnych intruzów z założenia będzie wysoce niedoskonała. Warto natomiast skoncentrować się na elemencie dobrze rozpoznanym, czyli samym realizowanym projekcie. Łatwiej jest opisać w pewnym wymiarze dobrze zdefiniowany projekt niż możliwe ledwie mgliście zarysowane zachowania potencjalnych intruzów.

W kontekście modelowania zagrożeń i analizy ryzyka dobrym podejściem wydaje się próba zdefiniowania możliwych do wystąpienia podatności realizowanego rozwiązania oraz, na zasadzie analogii, możliwych metod eksploatacji takich podatności. Pierwsze pytanie w modelowaniu zagrożeń brzmi: co jest zasadniczym przedmiotem projektu? Zagadnienie należy rozpatrywać w szerszej perspektywie. Powinna ona obejmować nie tylko architekturę naszego rozwiązania, ale również doprecyzować jego otoczenie oraz interakcje wewnętrzne i zewnętrzne.

Stąd dobrym punktem wyjścia do analizy poziomu bezpieczeństwa jest rozpoczęcie od stworzenia pewnej reprezentacji graficznej rozwiązania w postaci diagramu. Diagramu, który z jednej strony zobrazuje w sposób wykorzystywany na rzecz analizy bezpieczeństwa architekturę przedmiotowego rozwiązania, a zarazem wskaże interakcje zewnętrzne oraz

wpływ dostępności i poprawności działania elementów zewnętrznych. O ile teoretycznie rozpatruje się różne podejścia, w praktyce zawsze warto zacząć od bardzo ogólnego i stosunkowo prostego modelu i diagramu. Z założenia nie będzie on doskonaty, natomiast warto pamiętać, że żaden diagram nie odda w pełni precyzyjnie całej architektury i zasady działania rozwiązania.

Po wytworzeniu wersji inicjalnej diagramu, w trakcie pracy nad analizą podatności i zagrożeń, naturalnym jest iteracyjne pogłębianie, doprecyzowywanie tworzonyj reprezentacji graficznej. Diagram w trakcie prac nad modelem zagrożeń ulega nie tylko doprecyzowywaniu – ale i zmianom. Jest to proces naturalny. Zmiany są dowodem użyteczności przyjętego sposobu działania – gdyby nie narysowano poprzedniej, jak się okazuje pierwotnie niedoskonalej wersji architektury jako diagramu, nie byłoby możliwości korekty percepcji postrzegania analizowanej architektury.

Po narysowaniu diagramu zaczyna się uzupełniać katalog zagrożeń. Zazwyczaj osoba rysująca ma już na samym początku kilka z takich zagrożeń na myśli. Ich źródłem mogą być podobieństwa wytwarzanego rozwiązania do analizowanych uprzednio pod względem modelowania zagrożeń rozwiązań podobnych, albo świadomość aktualnie popularnych i wykorzystywanych metod ataku systemów informatycznych. Okazuje się bowiem, że w przypadku cyberbezpieczeństwa również można mówić o pewnej sezonowości występowania różnych typów ataków. Zazwyczaj bieżąca moda na konkretny wektor ataku wynika z katalogu zidentyfikowanych w niedalekiej przeszłości podatności oraz dostępności do wykorzystania gotowych fragmentów oprogramowania możliwych do natychmiastowej eksploatacji znanych podatności.

Znalezienie kilku oczywistych i kilku nieoczywistych podatności prowadzi do kolejnego pytania, które współstanowi istotę modelowania zagrożeń: co można i co planuje się zrobić z każdym ze zidentyfikowanych zagrożeń? Bez wątpienia pierwszą czynnością, którą względem każdego ze zidentyfikowanych zagrożeń należy wykonać – jest odnotowanie w katalogu zagrożeń do dalszej analizy. Gromadzenie zestawienia można rozpocząć przy krótkim zestawieniu w postaci na przykład współdzielonego przez analityków zasobu (dokument, biała tablica itp.). Gdy katalog wydaje się dopełniony, jego zawartość stanowi wkład do systemu zagadnień projektowych dotyczących bezpieczeństwa, narzędzia wspierającego planowe, terminowe i częściowo zautomatyzowane rozwiązywanie zagadnień projektowych, typu: system zarządzania zleceniami (ang. ticketing system), czy system śledzenia postępów w realizacji projektów informatycznych, jak na przykład oprogramowanie Jira.

Zebrany katalog często na tym etapie obejmuje również jako przedmiot analizy wybrane elementy czy scenariusze, które dopiero w wyniku pogłębiającego przebadania okazują się nie być w rzeczywistości problematyczne z punktu widzenia bezpieczeństwa. Dotychczasowe uwzględnienie ich wśród analizowanych zagadnień nie było jednak błędem – umożliwione poddanie zagadnienia pod uwagę, dzięki czemu przedmiotowy scenariusz jest świadomie odrzucany z dalszej analizy, gdyż nie wymaga żadnych działań neutralizujących. Modelowanie zagrożeń obejmuje również właściwe uwzględnianie scenariuszy, których negatywny wpływ nie musi być jednoznacznie oczywisty. Jeżeli nie jest to rzeczywisty problem bezpieczeństwa, można taki fakt udokumentować odpowiednim komentarzem, a w wybranych sytuacjach można przykładowo napisać kod testowy wykazujący prawidłowe działanie środków zaradczych.

Ważne jest, by z założenia kwestiami bezpieczeństwa teleinformatycznego zarządzać równie kompleksowo jak kompleksowo zarządza się innymi wymiarami realizacji projektów i tworzenia produktów w danym przedsiębiorstwie. Zasadniczym efektem końcowym fazy modelowania zagrożeń a zarazem wkładem etapu w proces dostarczania klientom procesów i usług jest przede wszystkim sama lista zidentyfikowanych do dalszej ewaluacji zagrożeń. Poprzedzone identyfikacją zagrożeń zaplanowanie działań neutralizujących nie kończy procesu modelowania zagrożeń. Kolejnym zadaniem, które należy zrealizować jest możliwe rzetelna ocena jakości dotychczas przeprowadzonych działań.

Podczas sprawdzania, co mogło zostać opisane nieprawidłowo, ważne jest, by wyszukiwać zagrożenia w każdym jednym elemencie diagramu DFD lub każdej części diagramu, która znajduje się wewnątrz odpowiednich granic zaufania. Stąd kolejne pytanie, na które odpowiedź koniecznie trzeba zweryfikować, brzmi: czy dla każdego z elementów diagramu rzeczywiście przeanalizowano możliwość wystąpienie każdego typu zagrożenia, jakkolwiek prawdopodobnego w kontekście typu danego elementu diagramu i roli tego elementu w całym analizowanym systemie.

Dla każdego z wypisanych zagrożeń dokładność opisu jest na tym etapie mniej istotna niż sam fakt umieszczenia w zestawieniu niekorzystnego scenariusza, który mógłby zajść. Oczywiście są pewne granice upraszczania opisu – zagrożenie opisane zbyt ogólnie przeważnie uniemożliwia wdrożenie realnie skutecznych środków zaradczych. Przykładowo, zagrożenie opisane jako „kto może manipulować treścią pliku” jest istotnie mniej użyteczne niż opis „kto może manipulować treścią plików bazy danych w pomocniczym katalogu klienta i spowodować niecelowe wyświetlanie na stronie pierwotnie nieprzeznaczonych do wyświetlania treści”. Przy modelowaniu zagrożeń oczekiwane jest właściwie konkretne dodefinowanie charakteru i kontekstu zagrożenia. Co mogłoby się wydarzyć w scenariuszu przełamania zabezpieczeń i gdzie dokładnie w systemie taki scenariusz może wystąpić.

Warto wspomnieć, że modelownie zagrożeń może do późnego etapu - a czasem zupełnie - abstrahować od atrybucji źródeł ataków, czyli pomijać identyfikację potencjalnych rzeczywistych aktorów realizujących dany atak, o ile zaproponowane metody neutralizacji ataków danego typu są w stanie również abstrahować od takiej atrybucji bez wpływu na skuteczność metod przeciwdziałania. Przykładowo, wdrożenie do zapewnienia pewności transmisji szyfrowania kanału komunikacyjnego można uznać za skuteczne działania neutralizujące zagrożenie wycieku informacji podczas transmisji niezależnie od tego, kto próbowałby taką transmisję podsłuchiwać. Można zatem nie mieć pewności co do podmiotu, źródła takiej aktywności. Można nie mieć pewności co do celu takich działań.

Na tym etapie może się również zdarzyć, że po w miarę kompletnym zidentyfikowaniu charakteru zagrożenia nie widać jeszcze oczywistej ścieżki jego neutralizacji. Na tym etapie najważniejsze jest, by samo zagrożenie zostało chociaż zidentyfikowane i wstępnie zaewidencjonowane – stanie się wkładem do późniejszej analizy zagrożeń i metod ich przeciwdziałania.

Klasycznie modelowanie zagrożeń stanowi element bardziej pojemnego zagadnienia określonego mianem analizy ryzyka. Analiza ryzyka przeważnie kładzie silny nacisk na uwzględnianie prawdopodobieństwa wystąpienia czy inaczej materializacji analizowanych niebezpiecznych scenariuszy. Nie warto zajmować się zagadnieniami wysoce nieprawdopodobnymi, bo szansa ich wystąpienia jest znikoma. Z drugiej strony trzeba uważać, by nie wpaść w pułapkę nieuzasadnionego merytorycznie upraszczania, czyli próby klasyfikowania scenariuszy jako nieprawdopodobnych, w rzeczywistości tylko dlatego, że zespół identyfikujący zagrożenia choć jest zagrożeniami świadom, to nie ma wiedzy czy środków, które wymagane byłyby dla neutralizacji zagrożenia, a samo zagrożenie w rzeczywistości zgodnie z wiedzą zespołu jest jak najbardziej prawdopodobne.

## **STRIDE**

<https://bibliotekanauki.pl/articles/2068668.pdf>

<https://medium.com/@arielhacking/examples-of-stride-threats-for-payment-applications-87a0ad0c3a21>

<https://threat-modeling.com/the-ultimate-list-of-stride-threat-examples/>

### **Metodyka STRIDE w modelowaniu zagrożeń**

Model STRIDE wywodzi swoją nazwę z zestawienia różnych metod ataku na systemy informatyczne, katalog zagrożeń obejmuje:

1. Spoofing
2. Tampering
3. Repudiation
4. Information disclosure
5. Denial of Service
6. Elevation of Privilege

Mnemonik STRIDE ułatwia systematyczną, kompletną analizę potencjalnych zagrożeń na rzecz wybranych, bądź często wszystkich, elementów składowych diagramu DFD. Przyłożenie sześciu typów potencjalnych zagrożeń do danego elementu pozwala nam założyć, że element ten jako komponent badanego systemu został wyczerpująco przeanalizowany pod względem możliwych potencjalnych zagrożeń bezpieczeństwa. Poniżej rozwinięto opis poszczególnych składowych zestawienia typów zagrożeń stanowiącego model STRIDE.

#### **Spoofing (podszycie się, np. spoofing określonego serwera)**

Scenariusz określany mianem spoofing to sytuacja, gdy wystąpiło podszycie się pod czyjąś tożsamość, gdy nieprawidłowo zweryfikowano tożsamość. Zjawisko może dotyczyć naruszenia autentyczności źródła strony internetowej, która ładuje się w wyniku wybrania pewnego adresu URL. Własność autentyczności polega na tym, że nazwa odpowiada pewnym przyjętym przez użytkownika oczekiwaniom.

W sytuacji, gdy atakujący wykonuje spoofing, dostarcza podróbkę – treść nieautentyczną, niezakładaną – w miejsce treści prawdziwej.

#### **Tampering (manipulacja, np. manipulowanie plikiem)**

„T” w STRIDE oznacza tampering, czyli nieautoryzowaną modyfikację. Nieautoryzowana modyfikacja dotyczyć może zarówno zmiany zawartości pewnego pliku, jak i podmienienia części informacji w realizowanej komunikacji sieciowej.

W prawdziwym ataku typu *man-in-the-middle* pierwszym krokiem jest przekonanie jednego z komputerów na diagramie, że wskazany komputer jest najlepszym pośrednikiem sieciowym, routerem dla jego pakietów. Takie przekierowanie może być zaskakująco skuteczne w przejęciu kontroli nad ruchem sieciowym, umożliwiając nowemu pośrednikowi w komunikacji na odczyt, zmianę, odrzucanie lub tworzenie dowolnych pakietów stanowiących element przejętej komunikacji. Dzięki skutecznemu przekierowaniu atakujący nie musi znajdować się wewnątrz obszaru wzajemnego zaufania.

Rozwiązywanie problemu manipulacji jest relatywnie prostsze niż rozwiązywanie problemu ataku poprzez spoofing. W przypadku plików lokalnych uprawnienia systemu operacyjnego są solidne, zakładając, że zostały poprawnie skonfigurowane. Podobnie w chmurze należy korzystać z możliwości zabezpieczania, które zapewnia system. W przypadku komunikacji sieciowej należy używać kryptograficznej ochrony integralności transmisji, na przykład poprzez wykorzystanie protokołu TLS.

### **Repudiation (wyparcie się, np. odrzucenie zamówienia)**

„R” w STRIDE oznacza wyparcie się, odrzucenie. Odrzucenie ma odmienną specyfikę niż pozostałe typy zagrożeń stanowiące model STRIDE. Jest to rzadziej występujące określenie i oznacza zrzeczenie się, zaprzeczenie lub jakikolwiek sposób przekazania, że nie jest się za coś odpowiedzialnym.

Przykładem wyparcia się, odrzucenia jest stwierdzenie o treści „Nie otrzymałem takiej wiadomości” a także „Nie można stwierdzić, czy na pewno otrzymałem taką wiadomość”.

Przyczyną materializacji takiego zagrożenia może być brak wdrożonych rejestrów przesyłanych wiadomości albo braki takich dzienników systemowych za wybrane podokresy. W celu neutralizacji zagrożenia niezbędne jest prowadzenie niepodważalnych rejestrów, na przykład dzienników systemowych, których zawartość zostanie dostarczona jako kontrargument w sytuacji podważania przez uczestnika pewnej aktywności swojego udziału w tej aktywności

### **Ujawnianie informacji**

Litera „I” w modelu zagrożeń STRIDE oznacza ujawnianie informacji (ang. *information disclosure*).

W sieci najlepsza poufność pochodzi z kryptografii. W rzeczywistości kryptografia jest najlepszym sposobem na ochronę każdego sekretu, ale wtedy trzeba bytoby zarządzać ogromną liczbą kluczów, a to jest skomplikowane. TLS w większości przypadków zajmuje się zarządzaniem kluczami za użytkownika.

W ramach systemu łatwiejsze może być użycie uprawnień. Większość serwerów WWW umieszcza mechanizmy kontroli dostępu pomiędzy użytkownikami i plikami w serwerze [WWW](#). Ponieważ użytkownik loguje się do serwera WWW, nie może zalogować się do niego przez SSH. Wadą tego rozwiązania jest konieczność wyboru i zarządzania mechanizmem uprawnień.

Poufność może dotyczyć zarówno treści komunikacji, jak i informacji o samym fakcie komunikacji. Czasami dane osobowe lub informacje biznesowe mogą zostać ujawnione w logach, raportach diagnostycznych lub niewłaściwie chronionych zrzutach ekranu. Należy więc zadbać o to, aby wrażliwe informacje nie były dostępne w logach, plikach tymczasowych i innych miejscach, które pierwotnie nie są postrzegane jako repozytoria danych.

### **Odmowa świadczenia usługi**

„D” w STRIDE oznacza denial-of-service (odmowa usługi). Istnieją ataki typu denial-of-service (lub DoS) przeciwko procesorom, sieciom i pamięci masowej. Najprostsze ataki denial-of-service to po prostu *brute-force*. Przy dużej liczbie żądań obejrzenia danej reklamy sieć się przepiętnia – albo z powodu połączeń przychodzących, albo, co bardziej prawdopodobne, danych wychodzących.

Najprostszym sposobem obrony przed atakami typu denial-of-service jest obfitość zasobów, które są trudne do wyczerpania przez atakujących. Jest to również właściwe rozwiązanie do szybkiej obsługi dużej liczby klientów, choć jest oczywiście drogie. Obrona przed rozproszonymi atakami jest czymś, co najlepiej zrobić na poziomie sieci lub dostawcy chmury.

Obrona przed sprytnymi atakami wymaga profilowania aplikacji i wiedzy o tym, jak będą się one zachowywać. Podobnie jak ujawnianie informacji dotyczy bezpieczeństwa i prywatności, utrzymywanie dostępności systemów jest zarówno właściwością bezpieczeństwa, jak i niezawodności.

### Zwiększenie poziomu uprzywilejowania

„E” w STRIDE oznacza podniesienie przywilejów, czyli zmniejszenie zestawu ograniczeń stosowanych wobec konta użytkownika. Jeśli więc obecnie użytkownik ma ograniczenie do wysyłania pakietów sieciowych do serwera AD, może podnieść swoje uprawnienia do uruchamiania kodu lub nawet do poziomu uprawnień administratora.

Jeśli ktoś może coś zmienić, a jego reakcja jest taka, że nie powinien mieć możliwości tego zrobić, może to oznaczać atak typu *elevation of privilege*. Niektóre problemy związane z podnoszeniem przywilejów dotyczą zasobów chronionych tylko przez nieprzejrzystość, jak przykładowo udostępnienie panelu kontrolnego administratora serwera WWW pod nieoczywistą ścieżką dostępu.

Wiele innych ataków na podniesienie przywilejów dotyczy sposobu przetwarzania nieuprzywilejowanych danych wejściowych albo pomylenia ścieżki wykonywania kodu w danym programie. Przykładowo, atak SQL injection pozwala podnieść poziom uprawnienia od prostego użytkownika WWW aż do konta bazy danych, gdzie serwer WWW był tylko pośrednikiem i pozwolił, by niektóre dane były traktowane jako kod.

Podobnie, atak typu *cross-site scripting* może działać przez uruchamianie kodu w przeglądarce użytkownika, co z perspektywy napastnika jest podniesieniem przywilejów w kontekście tego użytkownika. Unikanie ataków typu *elevation of privilege* wymaga dobrze zaprojektowanego i wdrożonego systemu autoryzacji.

### Zakres stosowalności modelu

Nie wszystkie zagrożenia ujęte w modelu STRIDE mają zastosowanie do wszystkich typów elementów diagramów przepływu danych. Najwięcej typów zagrożeń, a zarazem największy wysiłek analityczny, związany jest z opisem procesów składowych jako elementów diagramu DFD.

Pełne zestawienie stosowalności analizy poszczególnych typów zagrożeń modelu STRIDE na rzecz różnych typów składowych architektury systemu informatycznego opisanej diagramem DFD zestawiono na rysunku 2.

ELEMENT	S	T	R	I	D	E
Interaktorzy	✓		✓			
Procesy	✓	✓	✓	✓	✓	✓
Magazyny danych		✓	?	✓	✓	
Przepływy danych		✓		✓	✓	

Rys. 2. Elementy STRIDE stosowane do poszczególnych typów elementów diagramów DFD

Fig. 2. STRIDE elements used for individual types of DFD diagram elements

### Narzędzia dedykowane do modelowania zagrożeń

Istnieją narzędzia informatyczne dedykowane do modelowania zagrożeń, a w szczególności wspomagające poprawne tworzenie diagramów przepływu danych. Najpopularniejsze ogólnodostępne pakiety to:

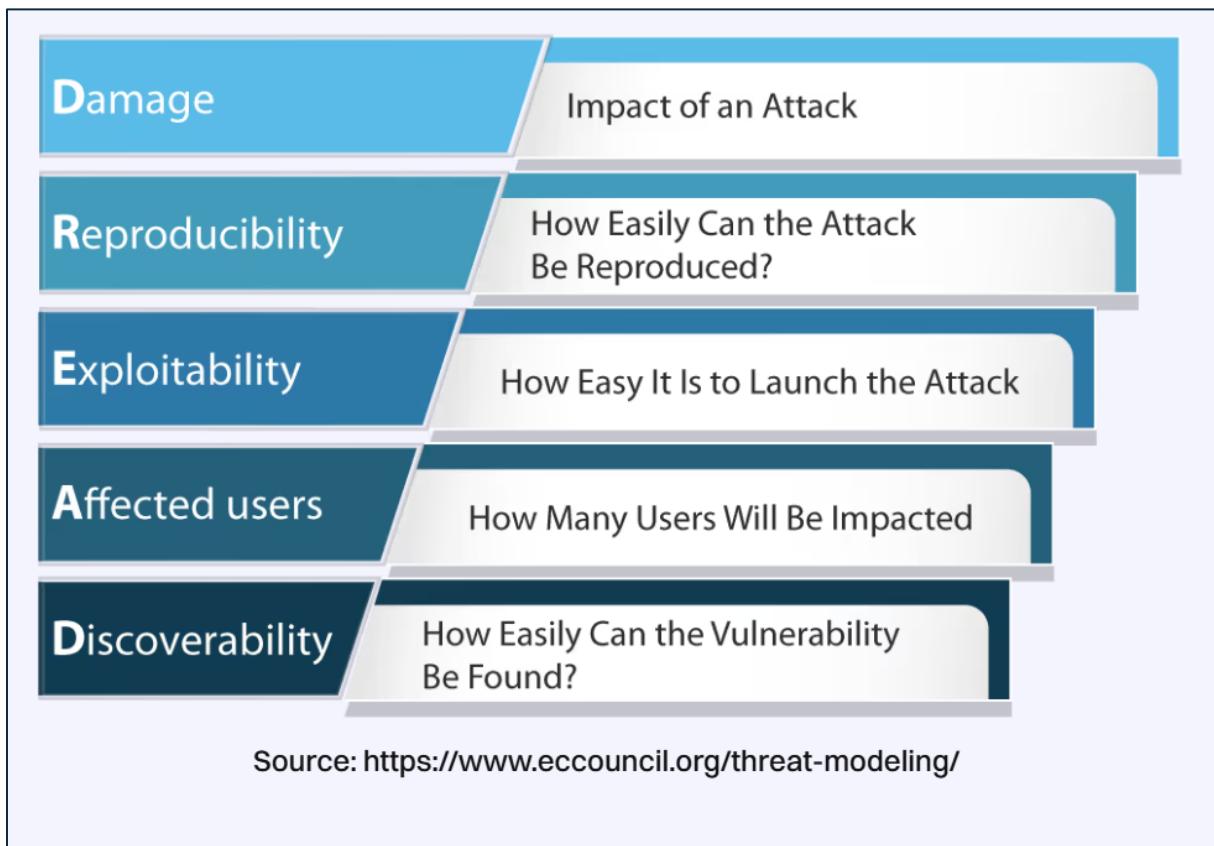
- Microsoft Threat Modeling Tool, <https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool>
- OWASP Threat Dragon, OWASP, <https://owasp.org/www-project-threat-dragon/>

## DREAD

[https://www.practical-devsecops.com/dread-threat-modeling/?srsltid=AfmBOooszEIT9zzR9Mw06cKInO8oC1geDbvQ2O9N2wWzy20r\\_k2VScL9](https://www.practical-devsecops.com/dread-threat-modeling/?srsltid=AfmBOooszEIT9zzR9Mw06cKInO8oC1geDbvQ2O9N2wWzy20r_k2VScL9)

<https://threat-modeling.com/dread-threat-modeling/>

DREAD to akronim od pięciu czynników oceny ryzyka: Damage Potential, Reproducibility, Exploitability, Affected Users i Discoverability. Metodologia ta polega na przyznaniu punktów od 0 do 10 dla każdego czynnika dla każdego zagrożenia i obliczeniu średniej arytmetycznej. Im wyższa jest średnia, tym wyższe jest ryzyko związane z zagrożeniem. Metodologia ta jest prosta w użyciu i pomaga w priorytetyzowaniu zagrożeń.



Model DREAD można porównać do Common Vulnerability Scoring System (CVSS) pod względem metodologii oceny powagi zidentyfikowanych zagrożeń. Firma Software Secured stosuje podejście mieszane, wykorzystujące zarówno DREAD, jak i CVSS przy ocenie podatności.

Framework DREAD może oceniać wagę pojedynczych zagrożeń, które zostały już zidentyfikowane przy użyciu innych metodologii, takich jak STRIDE. Gdy zagrożenie zostanie zidentyfikowane, DREAD pomaga zmierzyć jego potencjalną dotkliwość poprzez przypisanie ocen.

Jego metodologia może zapewnić szybki i skuteczny sposób identyfikacji i priorytetyzacji potencjalnych zagrożeń oraz pozwala organizacjom skupić się najpierw na najpoważniejszych zagrożeniach.

Jednak framework DREAD ma również pewne ograniczenia. Jednym z ograniczeń jest to, że koncentruje się wyłącznie na zagrożeniach technicznych i nie uwzględnia innych czynników, które mogłyby wpłynąć na wagę potencjalnego zagrożenia, takich jak wpływ na działalność biznesową lub reputację.

Dodatkowo, framework może nie zapewniać wystarczającego poziomu szczegółowości, aby w pełni ocenić wagę potencjalnego zagrożenia, a oceny przypisywane każdej kategorii mogą być subiektywne i różnić się w zależności od indywidualnych perspektyw.

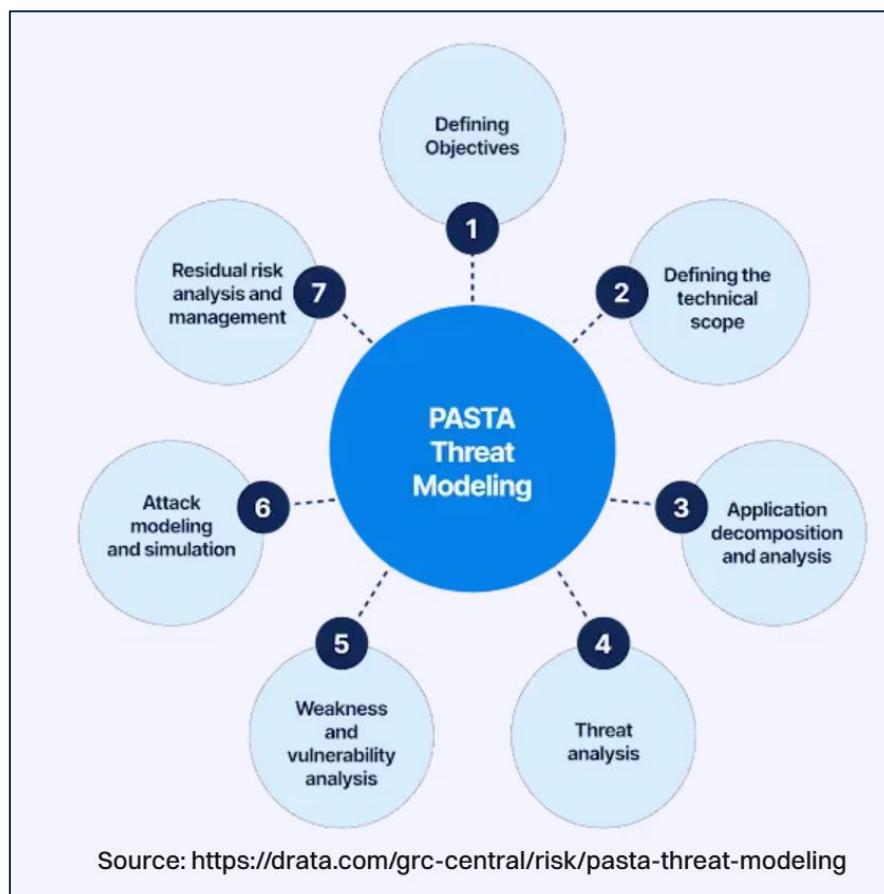
## PASTA

<https://www.broadcom.com/topics/threat-analysis>

PASTA oznacza *Process for Attack Simulation and Threat Analysis* (proces symulacji ataków i analizy zagrożeń). Jest to siedmioetapowa metodologia służąca do identyfikowania, analizowania i priorytetyzowania zagrożeń oraz ataków w aplikacjach software'owych. Framework PASTA jest kompleksowy i koncentruje się na podejściu do modelowania zagrożeń opartym na analizie ryzyka.

Metodologia PASTA stosuje siedmioetapowe podejście do modelowania zagrożeń:

1. **Define Objectives (Zdefiniuj cele):** Zidentyfikuj cele i założenia bezpieczeństwa systemu, który jest modelowany.
2. **Define Technical Scope (Zdefiniuj zakres techniczny):** Określ techniczny zakres systemu oraz jego granice.
3. **Decomposition and Analysis (Dekompozycja i analiza):** Rozbij system na mniejsze komponenty i przeanalizuj każdy z nich pod kątem potencjalnych zagrożeń.
4. **Threat Analysis (Analiza zagrożeń):** Zidentyfikuj i ustal priorytety potencjalnych zagrożeń oraz ich wektorów ataku.
5. **Vulnerabilities and Weaknesses Analysis (Analiza podatności i słabości):** Zidentyfikuj i przeanalizuj potencjalne podatności i słabości w systemie.
6. **Modeling and Simulation (Modelowanie i symulacja):** Stwórz wizualny model z wykorzystaniem diagramów i symulacji, aby ocenić poziom bezpieczeństwa systemu.
7. **Risk Impact Analysis (Analiza wpływu ryzyka):** Oceń ryzyka związane ze zidentyfikowanymi zagrożeniami i podatnościami oraz ustal ich priorytety pod kątem działań ograniczających ryzyko.

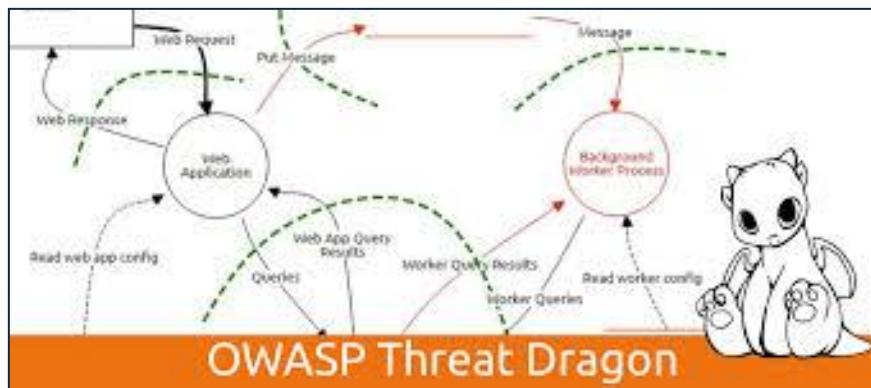


PASTA jest często stosowana w organizacjach z dojrzałym programem bezpieczeństwa. Może kierować opracowaniem środków zaradczych w celu rozwiązania zidentyfikowanych ryzyk. Ten framework jest elastyczny, pozwalając organizacjom dostosować metodologię do ich specyficznych potrzeb.

PASTA wymaga wysokiego poziomu wiedzy specjalistycznej, aby została poprawnie zaimplementowana, i jest zazwyczaj bardzo czasochłonna. Jest to również złożona metodologia, która może nie być odpowiednia dla mniejszych organizacji z ograniczonymi zasobami. Dodatkowo PASTA nie dostarcza szczegółowych wytycznych dotyczących sposobu adresowania zidentyfikowanych ryzyk, więc do opracowania skutecznego planu ograniczania ryzyka może być potrzebna dodatkowa wiedza ekspercka.

## OWASP Threat Dragon

OWASP Threat Dragon to darmowe, wieloplatformowe narzędzie typu open source do modelowania zagrożeń, służące do tworzenia diagramów przepływu danych (DFD) oraz identyfikowania potencjalnych zagrożeń bezpieczeństwa w aplikacjach i systemach oprogramowania. Jest to oficjalny projekt Fundacji OWASP, której celem jest poprawa bezpieczeństwa oprogramowania.



### Kluczowe funkcje i cel

- Tworzenie diagramów: Użytkownicy mogą tworzyć wizualne reprezentacje swoich systemów za pomocą diagramów przepływu danych (DFD), które pomagają wizualizować komponenty, przepływy danych i granice zaufania.
- Identyfikacja zagrożeń: Narzędzie wykorzystuje silnik reguł do automatycznego generowania listy potencjalnych zagrożeń i sugerowanych sposobów ich łagodzenia na podstawie elementów diagramu i wybranej metodologii.
- Obsługa metodologii: Obsługuje popularne metodologie i frameworki modelowania zagrożeń, w tym:
  - STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege).
  - LINDDUN (Linking, Identifying, Non-repudiation, Detecting, Undisclosed data, Non-compliance).
  - CIA (Poufność, integralność, dostępność).
  - PLOT4ai.
- Współpraca i integracja: Narzędzie wspiera podejście oparte na współpracy, umożliwiając zespołom programistycznym i ds. bezpieczeństwa wspólną pracę. Modele zagrożeń mogą być przechowywane w formacie tekstowym (np. JSON) i integrowane z cyklem życia tworzenia oprogramowania. Oferuje integrację z platformami takimi jak GitHub, GitLab i Bitbucket do przechowywania plików modeli.
- Raportowanie: Modele zagrożeń i związane z nimi zagrożenia można eksportować w formie raportów PDF w celach dokumentacyjnych i związanych z zgodnością.

### Opcje wdrażania

OWASP Threat Dragon jest dostępny w dwóch głównych wariantach:

- Aplikacja desktopowa: Instalowalna aplikacja zbudowana przy użyciu Electron dla systemów Windows, macOS i Linux. Zapisuje modele zagrożeń w lokalnym systemie plików.
- Aplikacja webowa: Wersja oparta na przeglądarce internetowej, którą można uruchomić w kontenerze Docker lub uzyskać do niej dostęp online, a która zazwyczaj przechowuje pliki w połączonym repozytorium GitHub.

**Materiały dodatkowe:**

- <https://github.com/hysnsec/awesome-threat-modelling?tab=readme-ov-file>
- <https://github.com/rhurlbut/CodeMash2019/blob/master/Robert-Hurlbut-CodeMash2019-Threat-Modeling-Workshop-20190108.pdf>
- <https://www.threatmodelingmanifesto.org/>
- <https://csrc.nist.gov/pubs/sp/800/154/ipd>
- <https://handouts.secappdev.org/handouts/2018/Seba%20Deleersnyder%20-%20Practical%20Threat%20Modeling.pdf>
- <https://45023718.fs1.hubspotusercontent-na1.net/hubfs/45023718/WC%20-%20PDFs/Threat%20Modeling%20Sample.pdf>
- <https://www.youtube.com/watch?v=rEnJYNkUde0>
- <https://www.youtube.com/watch?v=mL5G8HeI8zI>
- <https://simoneonsecurity.com/page/2/>
- <https://medium.com/@tahirbalarabe2/the-four-questions-of-threat-modeling-03a87e50c8bf>
- <https://wolski.pro/2019/12/modelowanie-zagrozen-teoretycznie/>
- <https://wolski.pro/2019/12/modelowanie-zagrozen-praktycznie/>
- <https://threat-modeling.com/the-ultimate-list-of-stride-threat-examples/>
- <https://github.com/bvoris/mitreatattackthreatmodeling>
- [https://www.youtube.com/watch?v=BhuFZ\\_4syVU](https://www.youtube.com/watch?v=BhuFZ_4syVU)

## Przykłady

### STRIDE – Przykład 1

#### Kontekst systemu (skrót)

- Aplikacja webowa sklepu internetowego.
- Użytkownik zakłada konto, loguje się (e-mail + hasło), przegląda produkty, dodaje do koszyka, opłaca zamówienie.
- Jest też panel administracyjny (zarządzanie produktami, zamówieniami, użytkownikami).

#### Zagrożenie 1 – Credential stuffing (podszywanie się pod użytkownika)

**Kategoria STRIDE:** S – *Spoofing*

#### Scenariusz:

Atakujący wykorzystuje bazę wyciekłych loginów i haseł z innych serwisów. Automatycznie testuje kombinacje e-mail/hasło na stronie logowania sklepu. Użytkownicy często używają tych samych haseł, więc część kont zostaje przejęta.

#### Przyczyny / słabości:

- Brak MFA (2FA).
- Brak limitu nieudanych prób logowania / brak CAPTCHA.
- Brak mechanizmów wykrywania nietypowej aktywności logowania (wiele prób z jednego IP / wielu IP na to samo konto).

#### Skutki:

- Przejęcie kont użytkowników.
- Możliwość składania zamówień na cudze dane, wgląd w historię zakupów, dane adresowe.
- Ryzyko obciążenia operatora płatności sporami / chargebackami.

#### Przykładowe środki zaradcze:

- Włączenie MFA.
- Ograniczanie liczby prób logowania, blokady czasowe, CAPTCHA.
- Monitorowanie anomalii logowań i blokowanie znanych list wyciekłych poświadczonych.
- Edukacja użytkowników + wymuszanie silnych, unikalnych haseł.

## Zagrożenie 2 – Modyfikacja danych użytkownika w bazie (np. SQL Injection)

Kategoria STRIDE: T – *Tampering*

### Scenariusz:

Formularz logowania (lub rejestracji) jest podatny na SQL Injection. Atakujący wstrzykuje złośliwe zapytanie, które modyfikuje rekordy w tabeli users (np. zmienia adres e-mail i hasło właściciela konta na dane atakującego).

### Przyczyny / słabości:

- Brak parametryzowanych zapytań / ORM.
- Brak walidacji i filtrowania danych wejściowych.
- Nadmierne uprawnienia konta DB (np. możliwość ALTER/UPDATE na całej bazie).

### Skutki:

- Modyfikacja kont użytkowników (przejęcie kont, blokowanie dostępu).
- Możliwość wstrzyknięcia backdoora, modyfikacji sald, rabatów.
- Potencjalne zniszczenie integralności danych (konieczność przywracania z backupu).

### Przykładowe środki zaradcze:

- Używanie *prepared statements* / ORM, unikanie konkatenacji SQL.
- Walidacja danych wejściowych po stronie serwera.
- Ograniczenie uprawnień kont DB (zasada najmniejszych uprawnień).
- Regularne testy bezpieczeństwa (pentesty, SAST/DAST).

### Zagrożenie 3 – Brak możliwości udowodnienia operacji (np. zwrotów)

Kategoria STRIDE: R – Repudiation

#### Scenariusz:

Użytkownik twierdzi, że **nie zlecał konkretnego zamówienia** ani zwrotu, chociaż w systemie widnieje, że było ono zrealizowane z jego konta. Aplikacja nie zapisuje wystarczających logów (brak IP, brak timestampów, brak identyfikatora operatora, gdy to był admin).

#### Przyczyny / słabości:

- Brak odpowiedniego logowania zdarzeń (kto, co, kiedy).
- Logi są łatwe do modyfikacji lub usunięcia (brak zabezpieczeń, brak centralnego systemu logowania).
- Brak powiązania operacji z konkretnym użytkownikiem / kontem / sesją.

#### Skutki:

- Spory z klientami (czy zamówienie było rzeczywiście złożone).
- Brak możliwości przeprowadzenia rzetelnego dochodzenia incydentu.
- Trudności dowodowe w przypadku postępowań prawnych.

#### Przykładowe środki zaradcze:

- Wprowadzenie szczegółowego audytu operacji (kto, co, kiedy, z jakiego IP).
- Wysyłanie potwierdzeń operacji (e-mail / SMS).
- Niezmienialne logi (WORM, centralny system SIEM).
- Ograniczony dostęp administratorów do logów (tylko odczyt, audit zmian).

#### **Zagrożenie 4 – Podsłuchanie danych logowania (brak HTTPS / złe TLS)**

**Kategoria STRIDE:** I – *Information Disclosure* (ale także T – Tampering)

##### **Scenariusz:**

Sklep działa bez HTTPS lub ma błędnią konfigurację TLS (słabe szyfry, mixed content). Użytkownik loguje się z niezabezpieczonej sieci Wi-Fi (np. kawiarnia). Atakujący w tej samej sieci przechwytuje ruch (sniffing) i odczytuje login, hasło lub cookie sesyjne.

##### **Przyczyny / słabości:**

- Brak wymuszania HTTPS (lub tylko na części serwisu).
- Brak HSTS, brak secure/HttpOnly na ciasteczkach sesyjnych.
- Niewłaściwa konfiguracja serwera (stare wersje TLS, słabe szyfry).

##### **Skutki:**

- Kradzież danych logowania i sesji.
- Możliwość pełnego przejęcia kont, dalsze ataki (np. na mail użytkownika, jeśli hasła są te same).
- Naruszenie ochrony danych osobowych (RODO itp.).

##### **Przykładowe środki zaradcze:**

- Wymuszenie HTTPS na całej stronie (redirect z HTTP do HTTPS).
- Poprawna konfiguracja TLS zgodna z aktualnymi rekomendacjami.
- Ustawianie Secure, HttpOnly, SameSite na cookie sesyjnych.
- HSTS, CSP, regularne testy konfiguracji (np. skanery SSL).

## Zagrożenie 5 – Atak DoS na mechanizm logowania

Kategoria STRIDE: D – *Denial of Service*

### Scenariusz:

Atakujący wysyła ogromną liczbę żądań do endpointu logowania (np. POST /login), powodując wysokie zużycie CPU/bazy danych. System zaczyna wolno działać lub całkowicie przestaje odpowiadać. Legalni użytkownicy nie mogą się zalogować ani zrobić zakupów.

### Przyczyny / słabości:

- Brak limitów zapytań (rate limiting) na wrażliwych endpointach.
- Brak filtrowania ruchu na poziomie WAF / firewall.
- Brak skalowania infrastruktury / mechanizmów auto-skalowania.

### Skutki:

- Niedostępność sklepu, utrata przychodów (szczególnie w godzinach szczytu / promocjach).
- Potencjalne koszty infrastruktury (jeśli aplikacja się skaluje bez kontroli).
- Utrata zaufania klientów („sklep ciągle nie działa”).

### Przykładowe środki zaradcze:

- Rate limiting i mechanizmy throttlingu.
- WAF, analiza i blokowanie anomalii w ruchu.
- Cache'owanie stron statycznych, oddzielenie warstwy logowania od reszty.
- Możliwość odcięcia / spowolnienia ruchu z wybranych IP / krajów.

## Zagrożenie 6 – Nieautoryzowany dostęp do panelu admina

Kategoria STRIDE: E – *Elevation of Privilege*

### Scenariusz:

Panel administracyjny sklepu jest dostępny pod przewidywalnym adresem (np. /admin). Logowanie admina chronione jest tak samo jak zwykłe konto użytkownika (brak MFA, brak dodatkowej weryfikacji). Dodatkowo backend nie sprawdza ról przy wywołaniu części endpointów – wystarczy mieć zalogowane konto, aby odpytać niektóre „adminowe” API.

### Przyczyny / słabości:

- Brak RBAC/ABAC – uprawnienia określone „na frontendzie”, a nie na backendzie.
- Brak MFA dla kont administracyjnych.
- Brak separacji środowisk (ten sam panel do administracji i obsługi klienta).

### Skutki:

- Atakujący, który przejął zwykłe konto, może eskalować uprawnienia do administratora (np. przez błąd w autoryzacji).
- Możliwość modyfikacji cen, rabatów, danych użytkowników, podmiany numeru konta bankowego, wstrzyknięcia złośliwego JS do frontendu.
- Pełne przejęcie sklepu (i danych klientów).

### Przykładowe środki zaradcze:

- Twarda autoryzacja po stronie backendu (sprawdzanie ról / uprawnień przy każdym żądaniu).
- MFA + dodatkowe zabezpieczenia dla kont admina (np. logowanie tylko z VPN / określonych IP).
- Ukrycie panelu admina (security by obscurity nie zastąpi zabezpieczeń, ale utrudni skanowanie).
- Zasada najmniejszych uprawnień dla adminów (podział ról – support, content, full admin).

## DREAD – Przykład 1

### Kontekst

**System:** sklep internetowy z logowaniem użytkownika (e-mail + hasło).

**Zagrożenie:** atak **credential stuffing** – atakujący używa listy wyciekłych loginów/haseł z innych serwisów i automatycznie testuje je na stronie logowania sklepu. Jeśli użytkownik ma to samo hasło, konto zostaje przejęte.

### Analiza DREAD

Skala: **1–10**, gdzie 1 = mało groźne, 10 = bardzo groźne.

#### 1. D – Damage potential (potencjalne szkody)

**Pytanie:** Jak duże szkody może wyrządzić udany atak?

- Atakujący ma pełny dostęp do konta użytkownika.
- Może:
  - złożyć zamówienia na cudze dane,
  - zmienić adres dostawy na swój,
  - podejrzeć dane osobowe, historię zamówień, może dane dot. zdrowia/zainteresowań (w zależności od sklepu),
  - zmienić hasło i zablokować prawowitemu właścielowi dostęp.
- Jeżeli konto jest powiązane np. z kartą zapisanej płatności / portfelem – potencjalne **straty finansowe**.
- Dochodzą szkody reputacyjne sklepu („straciliście moje konto/dane/pieniądze”).

**Ocena:**

**9/10** – może nie całkowita katastrofa dla całej firmy (nie ginie cała baza), ale dla pojedynczych użytkowników jest to bardzo poważne.

#### 2. R – Reproducibility (powtarzalność)

**Pytanie:** Jak łatwo, po jednorazowym sukcesie, powtórzyć atak?

- Jeśli jedna para login/hasło zadziała, nic nie stoi na przeszkodzie, żeby użyć tego samego narzędzia/skryptu dla tysięcy kolejnych użytkowników.
- Skrypt jest prosty: lista loginów/haseł + pętla wysyłająca żądania POST /login.
- Atak nie wymaga interakcji z użytkownikiem (jak phishing), wszystko dzieje się po stronie atakującego.
- Jeśli system nie ma limitów prób, to atak jest dosłownie „przepuszczeniem listy przez API”.

**Ocena:**

**8/10** – bardzo wysoka, jedyny „hamulec” to ewentualne limity / blokady, jeśli istnieją.

### 3. E – Exploitability (łatwość wykorzystania)

**Pytanie:** Jak trudno technicznie jest przeprowadzić atak?

- Narzędzia do credential stuffing są dostępne publicznie (lub łatwe do napisania samemu).
- Potrzebna jest lista wyciekłych poświadczeń – takie bazy krążą w sieci, w tym w darmowych formach.
- Wymagana wiedza techniczna: **podstawowy poziom** (umiejętność obsługi narzędzia/skryptu, znajomość endpointu logowania).
- Brak konieczności omijania skomplikowanych zabezpieczeń, jeśli:
  - nie ma MFA,
  - nie ma CAPTCHA,
  - nie ma sensownego rate limiting.

**Ocena:**

**8/10** – stosunkowo łatwo, zakładając, że zabezpieczenia są przeciętne lub słabe.

### 4. A – Affected Users (zasiąg wpływu)

**Pytanie:** Jak duża część użytkowników może zostać dotknięta?

- Dotknięci będą głównie ci, którzy:
  - używają **tych samych haseł** w wielu serwisach,
  - występują na listach wycieków,
  - nie mają MFA włączonego.
- Statystycznie niemały procent użytkowników używa tych samych haseł wszędzie – w realnym świecie to często kilkadziesiąt procent.
- Jeśli sklep ma np. 100k użytkowników, to realnie celem ataku mogą być **tysiące kont** (zależnie od jakości listy i nawyków użytkowników).

**Ocena:**

**7/10** – niekoniecznie 100% bazy, ale grupa może być duża, a do tego atak jest skalowalny.

### 5. D – Discoverability (wykrywalność luk)

**Pytanie:** Jak łatwo atakujący może odkryć, że system jest podatny / że atak zadziała?

- Sam fakt, że aplikacja ma formularz logowania → już jest potencjalnym celem credential stuffing.
- Atakujący zwykle **nie musi sprawdzać, czy system jest „podatny”** – po prostu próbuje:
  - jeśli część loginów/haseł działa → bingo,
  - jeśli wszędzie MFA, mocne limity, dziwne błędy → odpuszcza.
- „Luka” tutaj to raczej **brak zabezpieczeń**, a nie ukryty błąd logiczny – czyli z perspektywy atakującego:
  - łatwo sprawdzić,
  - łatwo ocenić skuteczność (konta, do których udało się zalogować).

**Ocena:**

**8/10** – wystarczy spróbować automatycznego logowania; nie trzeba wiedzieć o wewnętrznej architekturze.

#### **Podsumowanie punktacji DREAD**

Kryterium	Ocena (1-10)	Uzasadnienie (skrót)
D – Damage	9	Przejęcie kont, straty finansowe, dane osobowe, szkody reputacyjne.
R – Reproduc.	8	Łatwo powtarzalny atak na wielu użytkowników tym samym narzędziem.
E – Exploit.	8	Proste narzędzia, niski próg techniczny, dostępne listy haset.
A – Affected	7	Znacząca część bazy (ci, co recyklingują hasła).
D – Discover.	8	Wystarczy spróbować logowania; brak zabezpieczeń widać od razu.

**Łączny wynik:**

$$((9 + 8 + 8 + 7 + 8) / 5 = 40 / 5 = 8,0)$$

**Ocena ogólna: ~8/10 – wysoki priorytet ryzyka.**

To zagrożenie powinno znaleźć się w top listy do obsługi (wprowadzenie MFA, limitów, monitoringu).

## **STRIDE – Przykład 2**

### **Kontekst systemu (API zamówień)**

- Sklep internetowy udostępnia REST API:
  - GET /api/orders/{orderId} – pobranie szczegółów zamówienia,
  - PUT /api/orders/{orderId} – modyfikacja zamówienia (np. adres, status),
  - DELETE /api/orders/{orderId} – anulowanie zamówienia.
- API jest wykorzystywane przez frontend (SPA / aplikacja mobilna) oraz panel administracyjny.

### **Zagrożenie 1 – Podglądzanie cudzych zamówień (IDOR)**

#### **Kategoria STRIDE: I – *Information Disclosure***

##### **Scenariusz:**

API nie sprawdza, czy aktualnie zalogowany użytkownik jest właścicielem zamówienia. Atakujący po zalogowaniu iteruje orderId (np. 1001, 1002, 1003...) i pobiera szczegóły zamówień innych klientów.

##### **Przyczyny / słabości:**

- Brak kontroli dostępu na poziomie obiektów (*Broken Object Level Authorization*).
- Przewidywalne identyfikatory zamówień (kolejne liczby).
- Logika autoryzacji częściowo przeniesiona na frontend (np. ukrywanie przycisków zamiast weryfikacji po stronie API).

##### **Skutki:**

- Ujawnienie danych osobowych klientów (imię, nazwisko, adres, telefon, e-mail).
- Ujawnienie danych o zakupach (co, kiedy, za ile – wrażliwe np. przy lekach, sprzęt medyczny).
- Naruszenie przepisów o ochronie danych (RODO itp.), kary finansowe, utrata reputacji.

##### **Przykładowe środki zaradcze:**

- Weryfikacja w API: „czy użytkownik X jest właścicielem zamówienia Y?”.
- Używanie trudnych do odgadnięcia identyfikatorów (np. UUID).
- Minimalizacja zakresu zwracanych danych.
- Testy bezpieczeństwa API (w tym pod kątem IDOR).

## Zagrożenie 2 – Modyfikacja cudzych zamówień

Kategoria STRIDE: T – *Tampering*

### Scenariusz:

Endpoint PUT /api/orders/{orderId} pozwala modyfikować zamówienie (np. adres dostawy, metodę płatności, status). API nie sprawdza, czy użytkownik jest właścicielem zamówienia, ani czy ma odpowiednią rolę. Atakujący:

- loguje się na swoje konto,
- w narzędziach typu DevTools/Proxy zmienia orderId na cudze,
- wysyła żądanie z innym adresem, statusem, kwotą itp.

### Przyczyny / słabości:

- Brak autoryzacji na poziomie obiektu (jak wyżej).
- Brak rozróżnienia operacji użytkownika i admina (ten sam endpoint, brak ról).
- Słaba walidacja danych wejściowych (np. można zmienić price lub status na dowolne wartości).

### Skutki:

- Zmiana adresu dostawy – przekierowanie przesyłki do atakującego.
- Zmiana statusu z „opłacone” na „anulowane” (lub odwrotnie) – bałagan finansowo-księgowy.
- Możliwość manipulowania rabatami, kwotami, kosztami wysyłki.

### Przykładowe środki zaradcze:

- Wyraźne rozdzielenie API użytkownika i admina (inne endpointy / autoryzacja).
- Kontrola ról: zwykły user może zmieniać tylko swoje zamówienie i tylko określone pola (np. adres, NIE cena).
- Walidacja dopuszczalnych zmian statusów (workflow).
- Logowanie każdej modyfikacji (kto, co, kiedy).

### **Zagrożenie 3 – Podszywanie się pod innego użytkownika poprzez token / cookie**

**Kategoria STRIDE:** S – *Spoofing*

#### **Scenariusz:**

- Sesje użytkowników są utrzymywane za pomocą cookie z identyfikatorem sesji lub tokenem JWT.
- Cookie nie ma flag HttpOnly / Secure, token JWT można łatwo przechwycić (XSS, brak HTTPS).
- Atakujący kradnie token/ciasteczko sesyjne ofiary i używa go do wykonywania żądań do /api/orders/{orderId} jak prawowity właściciel.

#### **Przyczyny / słabości:**

- Brak odpowiednich flag zabezpieczających cookie.
- Brak HTTPS lub mieszanie treści HTTP/HTTPS.
- Brak ochrony przed XSS (CSP, walidacja danych wyświetlanych).
- Token JWT z długim czasem życia, brak możliwości szybkiego unieważnienia.

#### **Skutki:**

- Atakujący może czytać i modyfikować zamówienia ofiary, składać nowe, podglądać dane.
- Możliwość łańcuchowych ataków – reset haseł, zmiana adresu e-mail, itp.
- Naruszenie poufności i integralności danych.

#### **Przykładowe środki zaradcze:**

- HttpOnly, Secure, SameSite dla cookie sesyjnych.
- Wymuszenie HTTPS, poprawna konfiguracja TLS.
- Ochrona przed XSS (CSP, encoding, walidacja).
- Krótkie TTL tokenów, możliwość ich unieważniania (blacklist/rotation).
- Dodatkowa weryfikacja wrażliwych operacji (np. zmiana adresu, metoda płatności).

#### **Zagrożenie 4 – Brak rozliczalności operacji na zamówieniach**

**Kategoria STRIDE:** R – *Repudiation*

**Scenariusz:**

- System loguje jedynie „status zamówienia = ZMIENIONY”, bez informacji kto, co i kiedy zmienił.
- Administrator zmienia adres dostawy lub status zamówienia, ale później twierdzi, że tego nie zrobił.
- Użytkownik twierdzi, że nie anulował zamówienia, chociaż w systemie widnieje status „canceled”.

**Przyczyny / słabości:**

- Brak szczegółowego audytu API (brak informacji o użytkowniku, IP, payloadzie, czasie).
- Logi można łatwo modyfikować / usuwać (np. są zapisywane w tej samej bazie bez kontroli).
- Brak powiązania operacji z konkretną tożsamością (np. request zrobiony „przez system”, nie wiadomo jakiego użytkownika dotyczył).

**Skutki:**

- Trudności w wyjaśnianiu sporów z klientami („któro anulował zamówienie?”).
- Brak możliwości rzetelnego dochodzenia incydentów bezpieczeństwa.
- Problemy dowodowe w postępowaniach prawnych / audytach.

**Przykładowe środki zaradcze:**

- Pełny audit operacji w API: kto (ID użytkownika/rola), co (jaki endpoint, jakie parametry), kiedy, z jakiego IP.
- Logi wysypane do zewnętrznego, zabezpieczonego systemu (SIEM, WORM).
- Oddzielenie uprawnień do odczytu logów od uprawnień do operacji na systemie.
- Dodatkowe potwierdzenia (e-mail/SMS) przy wrażliwych zmianach.

## Zagrożenie 5 – Atak DoS na API zamówień

Kategoria STRIDE: D – *Denial of Service*

### Scenariusz:

- Endpoint GET /api/orders/{orderId} wykonuje ciężkie zapytania do bazy (łączenie wielu tabel, liczenie rabatów, historii płatności).
- Atakujący generuje ogromną liczbę żądań (np. z botnetu) z różnymi orderId.
- Serwer aplikacji i baza danych są przeciążone, inne żądania (również od legalnych użytkowników) zaczynają timeoutować.

### Przyczyny / słabości:

- Brak limitowania zapytań (rate limiting / throttling).
- Brak cache'owania odpowiedzi, nawet gdy dane zamówienia rzadko się zmieniają.
- Brak mechanizmów WAF / firewall przed API.
- Zbyt „ciężka” logika w jednym wywołaniu API.

### Skutki:

- Niedostępność funkcji przeglądania zamówień, panelu klienta i panelu admina.
- Klienci nie mogą sprawdzić statusu paczki, aktualizować danych, robić zakupów.
- Utrata przychodów i zaufania, szczególnie przy dużym ruchu (np. Black Friday).

### Przykładowe środki zaradcze:

- Rate limiting per IP / użytkownik / token.
- Cache odpowiedzi dla niezmiennych danych (np. zakończonych zamówień).
- WAF, mechanizmy anty-DDoS (np. na poziomie dostawcy chmury).
- Optymalizacja zapytań i możliwość degradacji funkcji (np. zwracanie uproszczonych danych przy wysokim obciążeniu).

## Zagrożenie 6 – Eskalacja uprawnień przez API (użytkownik → admin)

Kategoria STRIDE: E – *Elevation of Privilege*

### Scenariusz:

- Panel admina korzysta z tych samych endpointów co zwykły użytkownik, ale frontend *teoretycznie* ukrywa przyciski admina.
- Endpoint PUT /api/orders/{orderId} pozwala zmieniać pola, które powinny być dostępne tylko dla admina (np. status „shipped”, „refunded”).
- W praktyce backend nie sprawdza roli – jeśli zwykły użytkownik wyśle ręczne żądanie z odpowiednim body, API to przyjmie.

### Przyczyny / słabości:

- Brak centralnego systemu ról / uprawnień (RBAC).
- Oparcie się na logice frontendu („użytkownik tego nie zobaczy, więc nie wywoła”).
- Brak testów autoryzacji endpointów (np. brak prób wywołania adminowych endpointów z tokenem zwykłego usera).

### Skutki:

- Zwykły użytkownik może:
  - zmieniać status swoich (a czasem cudzych) zamówień,
  - anulować zamówienia po wysyłce,
  - markować zamówienia jako „opłacone” bez faktycznej płatności.
- Możliwość nadużyć finansowych, oszustw, chaos w logistyce.

### Przykładowe środki zaradcze:

- Wprowadzenie RBAC: każda operacja w API musi sprawdzać rolę użytkownika (user, support, admin itd.).
- Oddzielenie endpointów „user” od „admin” (różne adresy, różne polityki autoryzacji).
- Testy bezpieczeństwa nastawione na eskalację uprawnień (np. automatyczne skrypty próbujące admin calls z user tokenem).
- Zasada najmniejszych uprawnień – frontend nigdy nie decyduje sam o uprawnieniach.

## DREAD – Przykład 2

### Kontekst

**System:** sklep internetowy z API zamówień.

**Endpoint:** GET /api/orders/{orderId} (oraz potencjalnie PUT, DELETE).

### Zagrożenie:

Zastosowano przewidywalne identyfikatory zamówień (np. liczby całkowite), a API **nie sprawdza**, czy aktualnie zalogowany użytkownik jest właścicielem zamówienia.

Atakujący:

- loguje się na swoje konto,
- przechwytuje jedno poprawne żądanie do /api/orders/{orderId},
- zaczyna iterować orderId (1001, 1002, 1003...),
- pobiera dane **cudzych zamówień** – imię, nazwisko, adres, e-mail, szczegóły zamówień itp.

To klasyczny przypadek **Broken Object Level Authorization / IDOR**.

### Analiza DREAD

Skala 1–10 jak poprzednio.

#### 1. D – Damage potential (potencjalne szkody)

**Pytanie:** Jak duże szkody może wyrządzić udany atak?

- Atakujący uzyskuje dostęp do danych:
  - imię i nazwisko klienta,
  - adres dostawy (czasem domowy),
  - numer telefonu, e-mail,
  - historię zakupów (może być wrażliwa – np. leki, produkty medyczne, sprzęt intymny, książki o określonej tematyce).
- Dane te można wykorzystać do:
  - phishingu, ukierunkowanych ataków socjotechnicznych,
  - kradzieży tożsamości (łączenie danych z innymi wyciekami),
  - szantażu lub nękania (wrażliwe kategorie zakupów).
- Dla firmy:
  - naruszenie tajemnicy danych klientów,
  - ryzyko kar za naruszenie RODO / innych regulacji,
  - poważny cios w reputację („wyciekły moje dane i historia zakupów”).

### Ocena:

To jest bardzo poważne naruszenie poufności, często klasyfikowane jako incydent „data breach”. **9/10**

## **2. R – Reproducibility (powtarzalność)**

**Pytanie:** Jak łatwo powtórzyć atak po pierwszym sukcesie?

- Jeśli API choć raz zwróci dane cudzego zamówienia → atakujący wie, że:
  - identyfikatory są przewidywalne/iterowalne,
  - **brak kontroli własności obiektu.**
- Od tego momentu wystarczy:
  - pętla for po orderId od np. 1 do 1 000 000,
  - lub wielowątkowy skrypt/ narzędzie do fuzzingu.
- Żadne dodatkowe obchodzenie zabezpieczeń nie jest potrzebne – to „ciągnie się samo”, dopóki system nie wykryje anomalii.

**Ocena:**

Bardzo wysoka – jeden udany test pozwala w pełni zautomatyzować atak **9/10**

## **3. E – Exploitability (łatwość wykorzystania)**

**Pytanie:** Jak trudno technicznie jest przeprowadzić atak?

- Wymagane umiejętności:
  - umiejętność zalogowania się na konto i użycia narzędzi typu DevTools, Postman, Burp,
  - podstawowe zrozumienie REST API i parametrów w URL.
- Przykładowy sposób:
  1. Zalogować się w przeglądarce.
  2. Otworzyć DevTools → zakładka Network → przechwycić żądanie GET /api/orders/1234.
  3. Skopiować żądanie jako cURL lub do Postmana.
  4. Zmieniać ręcznie orderId (1233, 1232, 1235...) i obserwować odpowiedzi.
  5. Zautomatyzować to prostym skryptem.
- Nie trzeba być „hakerem z filmu” – wystarczy zainteresowany, średnio techniczny użytkownik.

**Ocena:**

Wymaga minimalnej wiedzy o API i użycia prostych narzędzi. **7/10**

#### 4. A – Affected Users (zasięg wpływu)

**Pytanie:** Ile użytkowników może zostać dotkniętych?

- Atak celuje w **całą bazę zamówień**, nie tylko w wybranych użytkowników.
- Jeśli identyfikatory są kolejne i atakujący iteruje po nich efektywnie:
  - może wyciągnąć dane **praktycznie wszystkich klientów**, którzy kiedykolwiek złożyli zamówienie,
  - również tych z przeszłości (dawne zamówienia nadal w bazie).
- Nie ma naturalnego ograniczenia zasięgu – jedynie limity to:
  - wydajność serwera,
  - limity ilości żądań (jeśli istnieją),
  - czas, jaki atakujący jest gotów poświęcić.

**Ocena:**

W praktyce może dotknąć większości lub wszystkich klientów → bardzo wysoka **9/10**

#### 5. D – Discoverability (wykrywalność luki przez atakującego)

**Pytanie:** Jak łatwo jest odkryć tę podatność?

- Trzeba:
  - zauważyc, że front odwołuje się do API /api/orders/{orderId},
  - spróbować zmienić orderId na inny i zobaczyć, czy API zwróci dane.
- Nie jest to „od razu oczywiste” dla zwykłego użytkownika, ale dla kogokolwiek, kto:
  - zna podstawy testowania bezpieczeństwa API,
  - lub świadomie „bawi się” w podmianę parametrów → to jest jedna z pierwszych rzeczy do sprawdzenia.
- Czyli: nie wynika od razu z samego interfejsu, ale jest względnie łatwo wykrywalne w trakcie prostego testu.

**Ocena:**

Ani super trivialne jak „spróbuj login=admin/admin”, ani bardzo ukryte – coś pośrodku.

**6/10**

#### Podsumowanie punktacji DREAD

Kryterium	Ocena (1–10)	Uzasadnienie (skrót)
D – Damage	9	Ujawnienie danych osobowych i historii zakupów – poważne naruszenie prywatności.
R – Reproduc.	9	Po jednym sukcesie atak można w pełni zautomatyzować i powtarzać masowo.
E – Exploit.	7	Wymaga tylko podstawowej wiedzy o API i prostych narzędzi.
A – Affected	9	Potencjalnie wszyscy klienci – cała baza zamówień.
D – Discover.	6	Wymaga aktywnego testowania, ale to standardowa technika dla kogoś technicznego.

**Łączny wynik:**

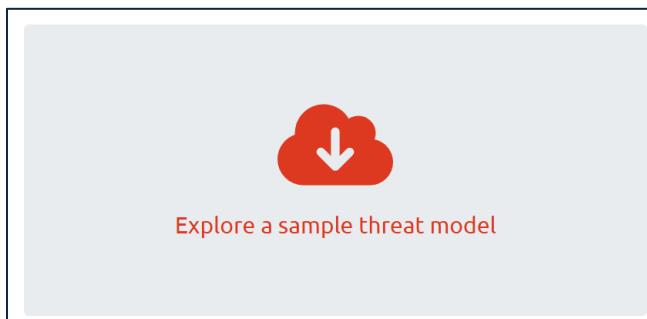
$$(9 + 9 + 7 + 9 + 6) / 5 = 40 / 5 = 8,0$$

**Ocena ogólna:** ~8/10 – bardzo wysokie ryzyko, krytyczne dla poufności danych.

## Zadania praktyczne

### Zadanie 1 – Narzędzia do modelowania zagrożeń

1. Zainstaluj lub uruchom w wersji webowej narzędzie OWASP Threat Dragon (link do dashboardu powyżej).
2. Otwórz przykładowe projekty/modeli dostarczone w narzędziu lub w dokumentacji i przeanalizuj je pod kątem:
  - zidentyfikowanych elementów systemu,
  - zdefiniowanych zagrożeń,
  - zastosowanych środków zaradczych.



Select a demo threat model from the list below

Demo Threat Model
Husky AI
Cryptocurrency Wallet
Generic CMS
IoT Device
Online Game
Payments Processing Platform
Renting Car Startup
Three Tier Web Application
New Blank Model

A screenshot of a dropdown menu titled "Select a demo threat model from the list below". The menu lists various threat models: Demo Threat Model, Husky AI, Cryptocurrency Wallet, Generic CMS, IoT Device, Online Game, Payments Processing Platform, Renting Car Startup, Three Tier Web Application, and New Blank Model. Each item is listed in a separate row with a thin vertical border.

3. Na podstawie jednego z przykładów przygotuj krótki opis modelu zagrożeń (co jest chronione, przed jakimi zagrożeniami i w jaki sposób).

Opcjonalnie: wykonaj analogiczną analizę, korzystając z narzędzia Microsoft Threat Modeling Tool (link poniżej) i porównaj sposób pracy obu narzędzi.

- <https://github.com/OWASP/threat-dragon>
- <https://www.threatdragon.com/#/dashboard>

Jak używać Owasp Threat Dragon:

- <https://www.threatdragon.com/docs/usage/getting-started.html>
- <https://medium.com/@iamblacklight/home-lab-threat-modeling-with-owasp-threat-dragon-f985be261597>
- <https://medium.com/@iamblacklight/threat-modeling-with-owasp-threat-dragon-part-2-d0196a9bb545>

Ewentualnie można wykorzystać narzędzie od Microsoft:

- <https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool>

## Zadanie 2 – Modelowanie zagrożeń na podstawie przykładów – Porównanie STRIDE i DREAD – DO SPRAWOZDANIA

### Materiały pomocnicze:

- [https://owasp.org/www-community/Threat\\_Modeling\\_Process](https://owasp.org/www-community/Threat_Modeling_Process)
- [https://cheatsheetseries.owasp.org/cheatsheets/Threat\\_Modeling\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html)
- <https://github.com/hysnsec/awesome-threat-modelling?tab=readme-ov-file>
- <https://www.diwebsity.com/2020/08/05/stride-threat-modelling/>

Wykonaj Threat modeling wybranego przez siebie przykładu np:

### Aplikacje webowe

- sklep internetowy,
- system rezerwacji wizyt (lekarz, fryzjer),
- panel klienta dostawcy internetu/energii,
- prosty serwis z logowaniem użytkowników.

### Aplikacje mobilne

- aplikacja bankowa,
- komunikator (chat),
- aplikacja fitness/zdrowotna zbierająca dane użytkownika.

### Systemy IoT / urządzenia sieciowe

- inteligentna żarówka/gniazdko sterowane z telefonu,
- kamera IP lub domowy monitoring,
- inteligentny domofon / zamek do drzwi,
- domowy router Wi-Fi.

### Usługi chmurowe / API

- publiczne API do obsługi zamówień,
- mikroserwis odpowiedzialny za logowanie użytkowników,
- prosty system przechowywania plików w chmurze

Możesz też wybrać inny, własny przykład ważne, aby dało się zidentyfikować elementy systemu, użytkowników oraz przepływ danych.

## 2. Opis systemu

Krótko opisz wybrany system (1–2 akapity), a następnie:

- wypisz głównych **aktorów** (użytkownicy, administratorzy, systemy zewnętrzne),
- wypisz kluczowe **zasoby** (np. dane klientów, płatności, logi, dane lokalizacyjne),
- opcjonalnie narysuj prosty **diagram** przepływu danych (DFD) lub architektury.

### 3. Threat modeling z wykorzystaniem STRIDE

Wykonaj modelowanie zagrożeń dla wybranego systemu, korzystając z framework'a **STRIDE**:

- dla najważniejszych elementów systemu (np. interfejs logowania, baza danych, API, urządzenie IoT)
- zidentyfikuj zagrożenia w każdej z kategorii STRIDE:
  - **Spoofing**,
  - **Tampering**,
  - **Repudiation**,
  - **Information Disclosure**,
  - **Denial of Service**,
  - **Elevation of Privilege**.

Dla każdego zagrożenia krótko opisz:

- na czym polega,
- jaki może mieć skutek dla systemu/użytkownika.

### 4. Ocena zagrożeń z wykorzystaniem DREAD

Dla zidentyfikowanych zagrożeń zastosuj framework **DREAD**:

- dla każdego zagrożenia oceń:
  - **Damage potential**,
  - **Reproducibility**,
  - **Exploitability**,
  - **Affected users**,
  - **Discoverability**,
- przypisz punktację (np. 1–10) i policz łączny wynik / średnią,
- na podstawie wyników ułóż listę zagrożeń wg **priorytetu** (od najpoważniejszych).

Możesz przedstawić to w formie tabeli.

### 5. Porównanie STRIDE i DREAD

Na końcu w sprawozdaniu:

- porównaj, jakie wnioski dał Ci **STRIDE**, a jakie **DREAD**,
- opisz **czym te frameworki się różnią**:
  - STRIDE – np. głównie do **identyfikacji typów zagrożeń**,
  - DREAD – np. głównie do **oceny ryzyka i priorytetyzacji** zagrożeń,
- wyjaśnij, **gdzie i jak** można je wykorzystywać (np. osobno oraz razem, np. najpierw STRIDE do znalezienia zagrożeń, potem DREAD do ich oceny).

**Zadanie 3 – Dodatkowe – Kurs od Microsoft i tryhackme (Dla chętnych)**

- <https://learn.microsoft.com/pl-pl/training/parts/tm-threat-modeling-fundamentals/>
- <https://tryhackme.com/room/threatemulationintro>
- <https://tryhackme.com/room/unifiedkillchain>