WOJSKOWA AKADEMIA TECHNICZNA

ROZPRAWA DOKTORSKA

mjr mgr inż. Artur STACHURSKI

MODEL I METODA PROGNOZOWANIA PODATNOŚCI OPROGRAMOWANIA NA ZAGROŻENIA W CYBERPRZESTRZENI

Promotor:

dr hab. inż. Andrzej NAJGEBAUER, prof. WAT

Promotor pomocniczy:

płk dr inż. Rafał KASPRZYK

Warszawa

Spis treści

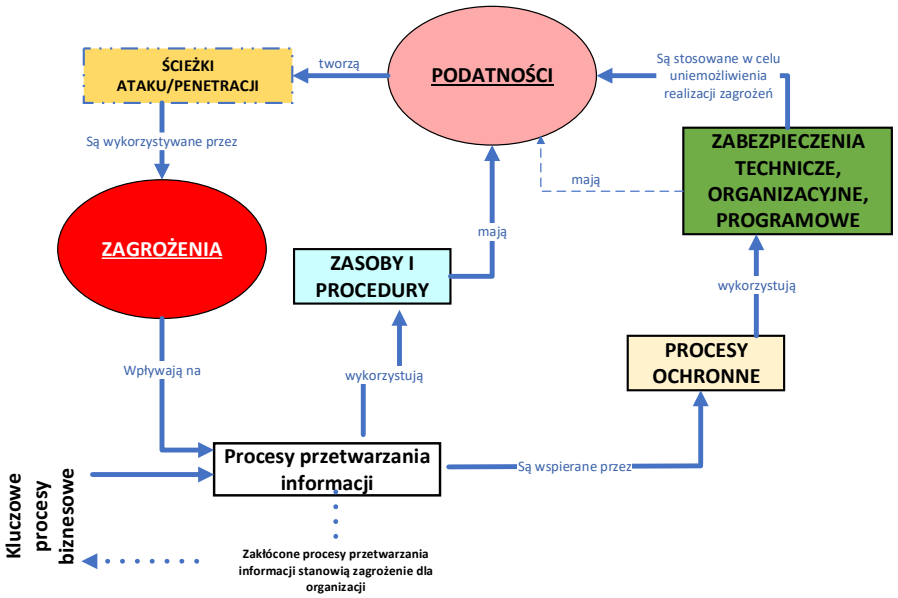
Wstęp

Wraz z postępem techniki komputerowej, rozwojem Internetu oraz rozpowszechnieniem się usług internetowych wciąż pojawiają się nowe zagrożenia i metody ataków w cyberprzestrzeni. W ciągu ostatnich dwudziestu lat, wymiana informacji na świecie stała się bardziej efektywna niż kiedykolwiek wcześniej. Ludzkość doświadczyła ogromnego skoku technologicznego, a gwałtowny rozwój nowoczesnych sieci transmisji danych, sieci telekomunikacyjnych oraz Internetu w znacznym stopniu wpłynął na nasz tryb życia, pracy oraz sposób spędzania wolnego czasu. Za pomocą kliknięcia myszy lub naciśnięcia ekranu smartfonu, odległości, które w nie tak odległej przeszłości były uważane za niemal nieosiągalne, zostają pokonywane w milisekundach. W wyniku tego łatwo, szybko i tanio możemy nawiązać kontakt z osobą znajdującą się na drugim końcu ziemi. Wraz z gwałtownym rozwojem technologicznym, wzrasta zatem liczba z globalnych powiązań występujących w cyberprzestrzeni, a wszelkie dziedziny naszego życia stają się uzależnione od technologii informacyjno-komunikacyjnych.

Nowoczesne odkrycia technologiczne pozwalają nam obecnie osiągnąć to, co w przeszłości było poza sferą marzeń, a nasz pogląd na wirtualny świat został na zawsze zmieniony. Niestety wraz z rozwojem możliwości, jakie oferuje nam sieć, zwiększyła się również liczba zagrożeń, które mogą dotyczyć wielu jej użytkowników. Ponadto, szeroko rozumiane powiązania, występujące w cyberprzestrzeni, prowadzą do uzależnienia wszelkich dziedzin życia od systemów informatycznych, a tym samym ich niezwykłej wrażliwości i podatności na cyberzagrożenia.

Współczesne wyzwania i zagrożenia bezpieczeństwa stanowią obecnie zasadniczą treść założeń większości strategii bezpieczeństwa państw demokratycznych oraz organizacji międzynarodowych. Podkreślają one fakt pojawienia się nowych wyzwań i zagrożeń, do których obecnie zaliczyć można przede wszystkim te, występujące w cyberprzestrzeni. Patrząc przez pryzmat obronności kraju, szczególnie groźne w skutkach mogą być nieuprawnione działania lub przejęcie kontroli nad elementami infrastruktury krytycznej państwa, a także kradzież czy niszczenie danych w systemach teleinformatycznych. Jednocześnie, rozważając problem zagrożeń dla indywidualnych użytkowników, mogą one doprowadzić do poniesienia strat finansowych, a nawet utraty zdrowia. Problem ten nie może być zlekceważony, dlatego też należy być świadomym istnienia zagrożeń występujących w cyberprzestrzeni jak również skutków, które te zagrożenia za sobą niosą. Szeroko rozumiane cyberbezpieczeństwo zależy dzisiaj w dużej mierze od możliwości

reagowania na incydenty zagrażające integralności cyberprzestrzeni i bezpieczeństwu informacji. Zapewnienie odpowiedniego poziomu bezpieczeństwa informacji wymusza konieczność ciągłego podnoszenia poziomu wiedzy, szybkiej identyfikacji zagrożeń, analizowania incydentów i wdrażania odpowiednich metod w celu zminimalizowania ryzyka potencjalnego ataku. Aby spełnienie tych wymagań było możliwe, konieczne jest posiadanie odpowiednich narzędzi informatycznych, które automatyzują wybrane procesy związane z zarządzaniem bezpieczeństwem IT.



Rys. 1. Zależności przyczynowo-skutkowe w procesie identyfikacji procesów, zagrożeń, podatności, zabezpieczeń [1]

W ostatnich latach miało miejsce wiele wydarzeń, które pokazały, jak bardzo istotne jest skuteczne zarządzanie incydentami i bezpieczeństwem systemów teleinformatycznych. Spośród grona bardzo spektakularnych ataków, a jednocześnie przynoszących ogromne straty wśród ofiar wymienić można m.in.:

- cyberataki przeprowadzone na infrastrukturę rządową Estonii w 2007 r. [2];
- ataki sabotażowe skierowane przeciwko irańskiemu programowi nuklearnemu przy wykorzystaniu zaawansowanego oprogramowania złośliwego: Stuxnet (2010) [3], Duqu (2011) [4], Flame (2012) [5];
- masowe cyberataki typu ransomware o nazwie WannaCry na m.in. instytucje rządowe, banki, firmy prywatne, przeprowadzane w celu wyłudzenia pieniędzy [6] i wiele więcej.

Analizując dotychczasowe strategie cyberataków, można zaobserwować, iż fundamentem każdego z nich jest pomyślne wykorzystanie przez atakującego co najmniej

jednej podatności oprogramowania na zagrożenia w cyberprzestrzeni (ang. *IT vulnerability*). Można ją rozumieć jako negatywne cechy oprogramowania, które pozwalają atakującemu naruszyć jego bezpieczeństwo. Z tego wynika, iż wiedza dotycząca szeroko rozumianych podatności na zagrożenia w cyberprzestrzeni pozwala zrozumieć w jaki sposób podmioty atakujące mogą zagrozić bezpieczeństwu eksploatowanych systemów informatycznych.

Prace naukowe skupione w obszarze związanym z badaniem podatności oprogramowania w większości przypadków skupiają się na podejściu jakościowym, polegającym na udoskonalaniu technik wykrywania podatności oraz zapobieganiu ich wystąpienia w oprogramowaniu. W przeciwieństwie do już wspomnianych, mniej miejsca w literaturze poświęcono na wykorzystanie metod ilościowych do badania podatności, które w sposób skuteczny i miarodajny znajdują zastosowanie w badaniu niezawodności oprogramowania (np. *Modele Wzrostu Niezawodności Oprogramowania - SRGM*).

Niniejsza praca stanowi próbę zaadoptowania i weryfikacji autorskich modeli matematycznych w procesie prognozowania podatności oprogramowania. Przedstawiono w niej następującą tezę:

> *Opracowanie modelu i metody prognozowania podatności oprogramowania na zagrożenia w cyberprzestrzeni pozwala na racjonalne kształtowanie poziomu bezpieczeństwa oprogramowania.*

Niniejsza teza zdeterminowała następujące zadania badawcze:

1. Opracowanie modelu matematycznego umożliwiającego szacowanie poziomu bezpieczeństwa oprogramowania w oparciu o wykrywanie podatności oprogramowania oraz zastosowanie działań profilaktycznych.
   1.1. Opracowanie metody wyznaczania intensywności wykrywania podatności oprogramowania na zagrożenia w cyberprzestrzeni (w danym przedziale czasu).
   1.2. Opracowanie metody szacującej skuteczność zastosowanych działań profilaktycznych.
2. Konstrukcja prototypu komputerowego narzędzia umożliwiającego analizę oraz prognozowanie podatności oprogramowania na zagrożenia w cyberprzestrzeni.
3. Weryfikacja opracowanego modelu i metody w oparciu o dane z bazy danych podatności oprogramowania na cyberzagrożenia.

Niniejsza praca została podzielona na pięć zasadniczych rozdziałów. W rozdziale pierwszym opisano charakterystykę obszaru badawczego. Przedstawiono definicje

podstawowych pojęć z zakresu cyberbezpieczeństwa, opisano cykl życia podatności, a także przedstawiono przykłady taksonomii oraz analizy ilościowej podatności. Elementy te pokazują, że podatności są istotnym zjawiskiem z punktu widzenia szeroko rozumianego cyberbezpieczeństwa.

Drugi rozdział obejmuje opis metod ilościowych analizy podatności, które zostały zidentyfikowane podczas przeglądu literatury. Przedstawiono opis metod służących do badania pojedynczej podatności – znanych jako systemy scoringowe, jak również opis modeli wykrywania podatności (VDM) oraz modeli predykcji podatności (VPM), dedykowanych do analizy określonych podzbiorów podatności, obejmujących wybrany pakiet oprogramowania lub jego moduły.

W rozdziale trzecim zawarto opis definicji podstawowych pojęć oraz narzędzi z zakresu probabilistyki i statystyki, które wykorzystane będą do konstrukcji modelu oraz zaproponowanych obliczeń. Następnie zaproponowano opis autorskich modeli do prognozowania podatności oprogramowania przy wykorzystaniu wybranych własności łańcucha Markowa. W oparciu o utworzone modele zdefiniowano oraz zaproponowano metody oszacowania i oceny niżej wymienionych właściwości:

- Poziom bezpieczeństwa oprogramowania;
- Intensywność wykrywania krytycznych podatności oprogramowania;
- Skuteczność zastosowanych działań profilaktyczno-naprawczych.

W dalszej części rozdziału trzeciego zawarto przykład obliczeń dla zaproponowanego procesu prognozowania podatności dla wybranej próbki pozyskanych danych do badań.

Rozdział czwarty, stanowi komplementarną część do opisanych modeli w rozdziale trzecim, i zawiera opis oraz przykład wykorzystania mechanizmów uczenia maszynowego w celu przyporządkowania nowo wykrytej podatności do zdefiniowanych wcześniej kategorii podatności. Do realizacji zadania klasyfikacyjnego wykorzystano i porównano wyniki dla czterech modeli uczenia maszynowego: regresji logistycznej, analizy dyskryminacyjnej liniowej, analizy dyskryminacyjnej kwadratowej oraz modelu wykorzystującego algorytm *k* najbliższych sąsiadów.

W rozdziale piątym zaprezentowano założenia oraz strukturę i funkcjonalność autorskiej aplikacji wspierającej obliczenia dla zaproponowanych modeli i metod. Całość pracy zwieńczona jest podsumowaniem, które zawiera wnioski z przeprowadzonych badań i analiz.

1. Charakterystyka obszaru badawczego

Niniejszy rozdział stanowi wprowadzenie do problematyki analizy podatności oprogramowania na zagrożenia w cyberprzestrzeni. W oparciu o przeprowadzony przegląd literatury dokonano usystematyzowania nazewnictwa oraz definicji podstawowych pojęć, którymi posługują się specjaliści od bezpieczeństwa informacyjnego, takich jak:

- cyberprzestrzeń,
- zagrożenie,
- incydent,
- atak,
- podatność,
- bezpieczeństwo informacji oraz bezpieczeństwo teleinformatyczne.

Następnie, scharakteryzowano cykl życia podatności, a także przedstawiono przykłady wybranych klasyfikacji podatności. W ostatnim podrozdziale zawarto wyniki przykładowej analizy ilościowej podatności w ujęciu statystycznym.

1.1 Definicja wybranych pojęć z dziedziny cyberbezpieczeństwa

1.1.1 Cyberprzestrzeń

Termin cyberprzestrzeń (ang. *cyberspace*) ukształtował się przez niezwykle intensywny rozwój wynalazku na przełomie XX i XXI wieku jakim jest Internet. Przyjmuje się, że po raz pierwszy użył go Wiliama Gibson, który w swojej powieści zatytułowanej „*Neuroromancer*" określił ją jako: „*(...) konsensualną, halucynację, doświadczaną każdego dnia przez miliardy uprawnionych użytkowników we wszystkich krajach, przez dzieci nauczane pojęć matematycznych. Graficzne odwzorowanie danych pobieranych z banków wszystkich komputerów świata. Niewyobrażalna złożoność (...)*." [7].

W 1996 roku Pierre Delvy w swoim raporcie „Deuxieme Deluqe", przygotowanym na zlecenie Komisji Kultury Rady Europy [8] określa cyberprzestrzeń jako wirtualną przestrzeń tworzoną przez komputery, które są połączone ze sobą w sieć oraz potrafią się komunikować. Patrząc przez pryzmat najnowszych osiągnięć technologicznych definicja ta uwzględnia wszystkie systemy komunikacji z udziałem komputerów, telefonów, urządzeń Internetu rzeczy (ang. *Internet of Things*) i sieci teleinformatycznych, w tym również sieci telefonii komórkowej. Z drugiej strony należy zaznacza się, aby nie mylić pojęcia cyberprzestrzeni z łączącą poszczególne terminale siecią komputerową.

Definicja zawarta w amerykańskich dyrektywach: *National Security Presidential Directive 54* oraz *Homeland Security Presidential Directive 23 (NSPD-54/HSPD23)* określa cyberprzestrzeń jako szeroko rozumianą infrastrukturę sieciową technologii informacyjnej (tzw. „nowe media"), która również obejmuje, poza Internetem, sieci telekomunikacyjne, systemy komputerowe oraz różnorodne procesory i kontrolery, ściśle powiązane z infrastrukturą krytyczną państwa (ang. *critical infrastructure*). Systemy te w nomenklaturze anglosaskiej są określane jako SCADA (ang. *Supervisory Control And Data Acquisition*), będące pochodną automatycznych systemów sterowania obiektami przemysłowymi. Istotne cechy, którymi można opisać cyberprzestrzeń to [9]:

- *niematerialny charakter,*
- *brak możliwości określenia granic,*
- *zdecentralizowanie,*
- *płynny i plastyczny charakter,*
- *powszechna dostępność,*
- *przetwarzanie i dokładne obliczanie w czasie rzeczywistym.*

W ujęciu matematycznym, model szkieletu cyberprzestrzeni można zdefiniować jako następujący wektor [10]:

$$CyberSpace(t) = \langle CNet(t), CAs(t), CTs(t), AMs(t), SMs(t) \rangle \tag{1.1}$$

gdzie:

$CNet(t)$ - model opisujący topologię i charakterystyki ilościowe sieci Internet (lub jej fragmentu będącego przedmiotem zainteresowania ze względu na cel modelowania);

$CAs(t)$ - aktorzy cyberprzestrzeni np. użytkownicy, administratorzy, hakerzy;

$CTs(t)$ - cyberzagrożenia występujące lub mogące potencjalnie wystąpić (np. sieci botnet, złośliwe oprogramowanie);

$AMs(t)$ - metody/mechanizmy ataków będące możliwymi realizacjami cyberzagrożeń (np. atak typu DDoS zrealizowany z wykorzystaniem cyberzagrożenia typu botnet);

$SMs(t)$ - metody/mechanizmy zabezpieczeń elementów składowych sieci Internet (np. instalacja oprogramowania antywirusowego lub firewall, systemy IDS/IPS).

Parametr $t \in T = \{1,2,3,\dots\}$ oznacza zdyskretyzowany czas, gdzie:
T– zbiór dyskretnych chwil.

Z powodu częstego braku spójności w definicji cyberprzestrzeni stanowi ona olbrzymie wyzwanie dla systemów prawnych wszystkich państw. Cyberprzestrzeń nie może być również pojmowana jako podmiot prawa, co powoduje, że nie istnieje żadna osoba bądź firma odpowiedzialna za to co się dzieje w sieci jako całości. Ponadto coraz częściej wykorzystywana jest do prowadzenia działalności przestępczej oraz terrorystycznej, a także została uznana – po lądzie, morzu, przestrzeni powietrznej i kosmicznej – za piąte środowisko walki [11].

Na potrzeby niniejszej pracy przyjęto następujące definicje cyberprzestrzeni oraz cyberprzestrzeni RP:

Definicja 1.1.1:

Cyberprzestrzeń *- cyfrową przestrzeń przetwarzania i wymiany informacji tworzoną przez systemy i sieci teleinformatyczne, wraz z powiązaniami pomiędzy nimi oraz relacjami z użytkownikami* [12].

Definicja 1.1.2

Cyberprzestrzeń Rzeczypospolitej Polskiej *– cyberprzestrzeń w obrębie terytorium Państwa Polskiego i w lokalizacjach poza jego terytorium, gdzie funkcjonują przedstawiciele RP (placówki dyplomatyczne, kontyngenty wojskowe)* [13].

Podsumowując, w kontekście nieustającego rozwoju technologii informacyjno-komunikacyjnych (ang. *Information and Communication Technologies, ICT*) można przypuszczać, że znaczenie cyberprzestrzeni będzie się ciągle zwiększać w wielu dziedzinach naszego życia i aktywności społecznej, czego skutkiem może być pojawienie się kolejnych zagrożeń dla jej użytkowników.

1.1.2 Zagrożenie

Na potrzeby niniejszej pracy wykorzystano definicję zagrożenia zbudowaną na podstawie definicji zaproponowanej w [14]:

Def 1.1.3

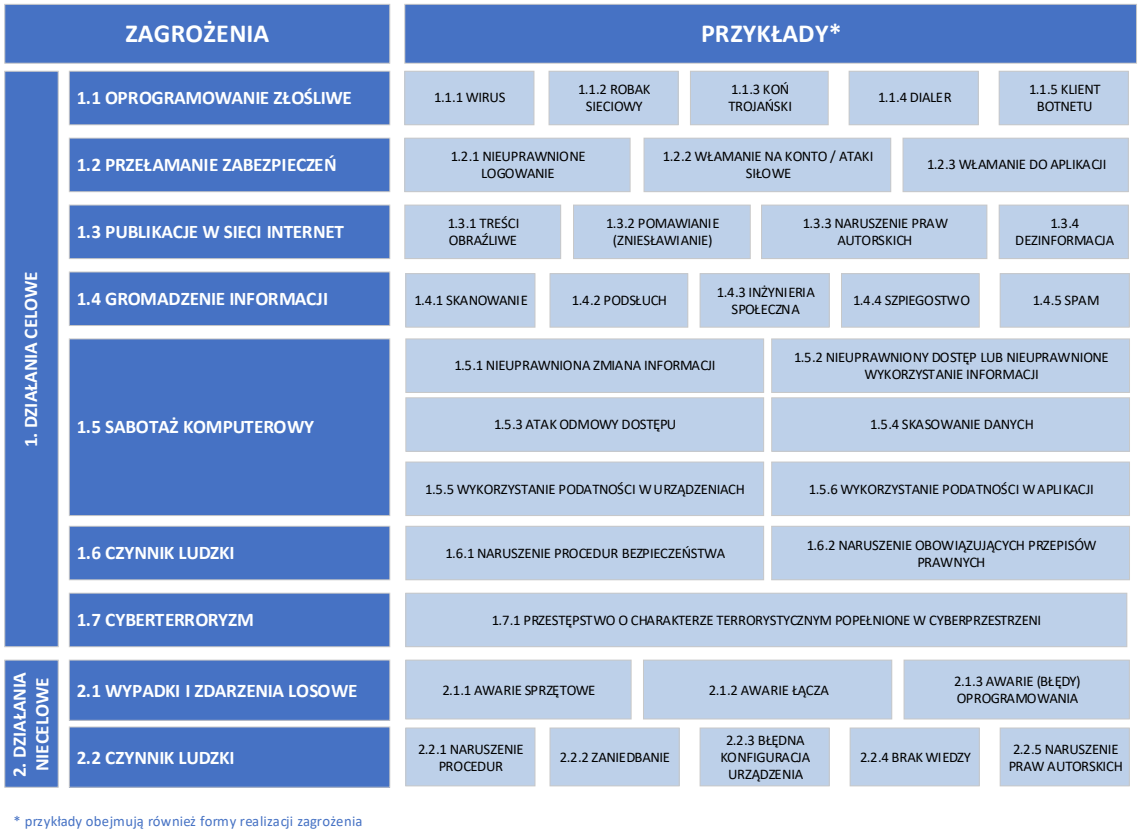
Zagrożenie *to potencjalne działania człowieka (lub zaniechanie takich działań) albo sił wyższych, dotyczące bezpośrednio zasobu teleinformatycznego lub organizacji procesu przetwarzania informacji i mogące (po wykorzystaniu podatności, jeśli istnieje)*

spowodować, w zależności od konkretnego atrybutu bezpieczeństwa: tajności, integralności lub dostępności, straty proporcjonalne do wagi procesu krytycznego, wspieranego przez ten proces i wykorzystywane w nim zasoby.

Aby zagrożenie spowodowało szkody po stronie ofiary (np. uszkodzenie sprzętu, kradzież danych, itd.), musi wystąpić jego tzw. „realizacja" na istniejącym zasobie (t. j. zasoby informacyjne, programowe, fizyczne, infrastrukturalne). Zasób ten jednocześnie musi posiadać podatność (definicja – patrz rozdział 1.1.5). Ciąg zdarzeń przedstawiający realizację zagrożenia nazywany jest zwykle scenariuszem realizacji zagrożenia, w którego wyniku powstaje incydent (z zakresu bezpieczeństwa informacyjnego), co zobrazowano na Rys. 2 [15].

Rys. 2. Schemat ilustrujący proces realizacji zagrożenia. Źródło: [15]

Informacje o zagrożeniach pochodzą z wielu źródeł. Jednakże pełne ich zdefiniowanie nie jest zadaniem trywialnym, gdyż problemem może być brak standardowych kategorii zagrożeń. Przykładowy model klasyfikacji zagrożeń występujących w cyberprzestrzeni, utworzony przez zespół CERT Polska (*Computer Emergency Response Team*) podczas wieloletnich obserwacji i praktyk, przedstawiony został na rysunku Rys. 3.



Rys. 3. Model klasyfikacji zagrożeń utworzony przez zespół CERT Polska. Źródło: [16]

1.1.3 Incydent

Jak wspomniano w poprzednim podrozdziale wynikiem realizacji zagrożenia jest incydent, który można zdefiniować jak poniżej [17]:

Def 1.1.4

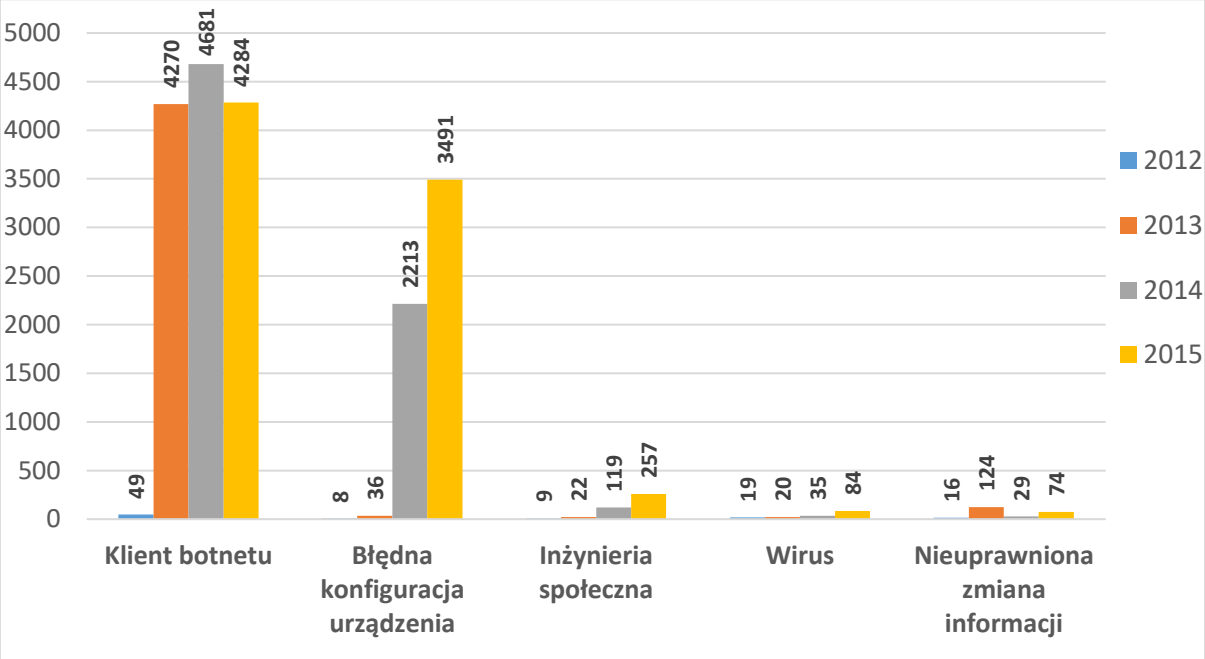
Incydent z zakresu bezpieczeństwa informacyjnego *to zdarzenie lub ciąg zdarzeń które powoduje lub może spowodować zmianę wartości atrybutów bezpieczeństwa informacji (tajności, integralności, dostępności lub innych) i które zwykle jest wynikiem:*

- *działania siły wyższej (w tym awarii i katastrof),*
- *działania osoby nieuprawnionej,*
- *lub błędnego działania osoby uprawnionej.*

Postępowanie z incydentami bezpieczeństwa informacji we współczesnych organizacjach stanowi bardzo ważny element kompleksowego systemu zarządzania [18]. Zarządzanie incydentami, wraz z procesami zarządzania ryzykiem (ang. *risk management*), zarządzania zdarzeniami bezpieczeństwa (ang. *security event management*), zarządzania

podatnościami (ang. *vulnerability management*) i testami bezpieczeństwa, ma na celu zapewnienie ciągłości operacyjnej oraz ograniczenie wpływu przypadków naruszeń bezpieczeństwa aktywów informacyjnych na działalność organizacji, w tym na ciągłość operacyjną jej procesów biznesowych, systemów i infrastruktury teleinformatycznej.

Wiele zespołów zajmujących się szeroko rozumianym cyberbezpieczeństwem podejmuje próby analizy zdarzeń występujących w cyberprzestrzeni w ujęciu liczbowym. Zespół CERT Polska gromadzi informacje o incydentach komputerowych, które wystąpiły w obszarze polskiej cyberprzestrzeni. Przykładową statystykę dotyczącą wybranych incydentów w cyberprzestrzeni, które zostały zidentyfikowane przez zespół CERT Polska przedstawiono na wykresie Rys. 4.



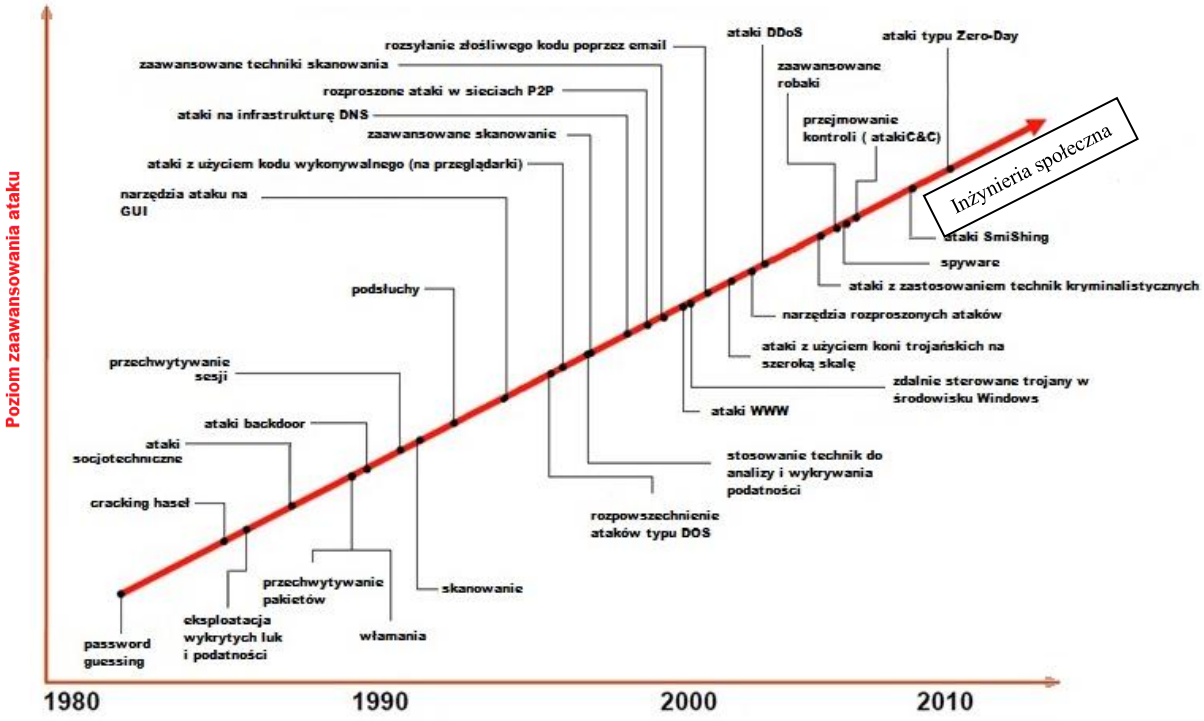
Rys. 4. Statystyka wybranych incydentów w cyberprzestrzeni RP w latach 2012-2015. Źródło: opracowanie własne na podstawie [19] [20] [21] [22]

1.1.4 Atak cybernetyczny

Jedną z form realizacji zagrożenia opisywanego w poprzednim podrozdziale (z klasy nieuprawnionych i przestępczych działań ludzi) jest atak, który można zdefiniować następująco [23]:

Def 1.1.5

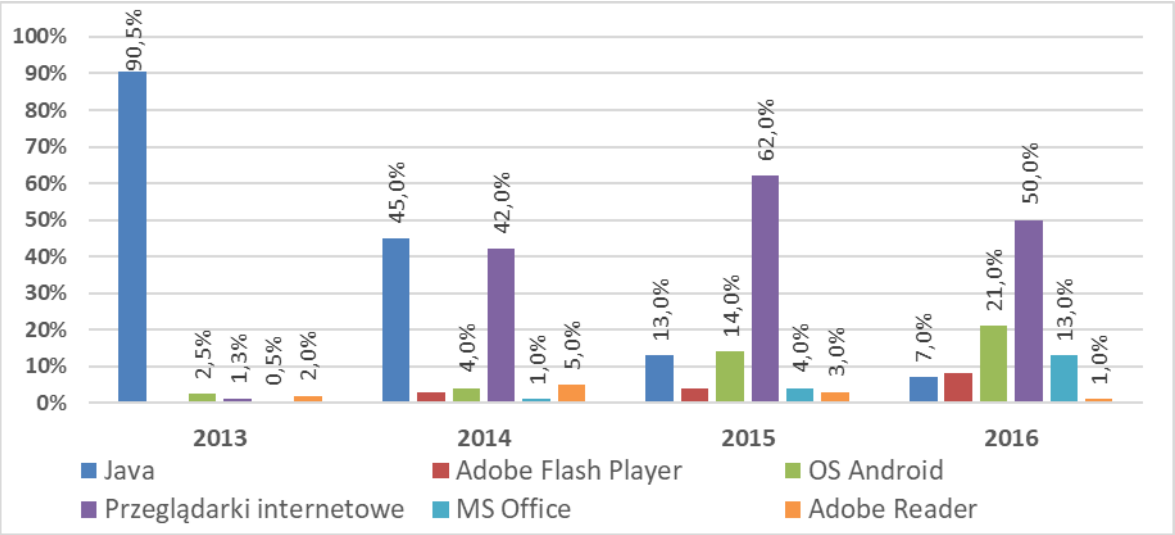
Atak *jest to próba zniszczenia, ujawnienia, zmiany, wyłączenia, kradzieży lub uzyskania nieautoryzowanego dostępu lub nieuprawnionego użycia zasobów, gdzie zasobami określa się wszystko, co ma wartość dla organizacji będącej ich właścicielem.*



Rys. 5. Ataki cybernetyczne na przełomie XX/XXI wieku. Źródło: Opracowanie własne

Jednocześnie, do przeprowadzenia ataku niezbędne są narzędzia, zmieniające się w zależności od rodzaju ataku. Mowa tu m. in. o złośliwym oprogramowaniu, technikach inżynierii społecznej (ang. s*ocial engineering*), czy też możliwościach uzyskania fizycznego dostępu do zasobów systemu. Warto zaznaczyć, iż pojęcie ataku nie jest równoznaczne z pojęciem zagrożenia, a niepoprawne utożsamianie może doprowadzić do błędów w budowie systemu ochrony zasobów informacyjnych [15].

Wiedzę dotyczącą statystyk ataków cybernetycznych można pozyskać między innymi z wyników badań przeprowadzanych w laboratoriach największych firm oferujących oprogramowanie antywirusowe. Statystyki te pokazują, że na przestrzeni ostatnich lat sposoby wykorzystywane do przeprowadzenia ataku ulegały zmianom [19] [20] [21] [22] [24] [25] [26] [27]. Mowa tutaj o powstaniu wielu nowych, a jednocześnie ewolucji wcześniej występujących metod ataku, które jak wskazują analizy z przeprowadzonych ataków cybernetycznych stają się coraz bardziej zaawansowane pod względem technologicznym, przez co uznawane są za trudniejsze do zwalczenia lub zneutralizowania. Przykładowa statystyka ilustrująca udział procentowy oprogramowania, którego podatności wykorzystano do przeprowadzenia ataków cybernetycznych w latach 2013-2016 została przedstawiona na rysunku Rys. 6.



Rys. 6. Udział procentowy oprogramowania, którego podatności wykorzystano do przeprowadzenia ataku cybernetycznego w latach 2013-2016 z danych organizacji Kaspersky Lab. Źródło: [24] [25] [26] [27]

1.1.5 Podatność

W literaturze przedmiotu znaleźć można wiele prób definicji podatności (ang. *IT vulnerability*, *vulnerability*). Definicja ISO 27005:2008 dotycząca zarządzania ryzykiem w systemie bezpieczeństwa informacji określa podatność systemu na cyberzagrożenia jako wadę poszczególnych jego elementów lub ich zbioru, które mogą być wykorzystane przez jedno lub więcej zagrożeń, gdzie elementem systemu jest wszystko, co może mieć wartość dla organizacji, jej działalności gospodarczej, w tym zasobów informacyjnych, które wspierają cel jej działania [28]. Norma ta wskazuje, iż podatności mogą dotyczyć wielu czynników wpływających na działanie systemu, zarówno związanych z jego funkcjami, konfiguracją, obsługą czy miejscem eksploatacji, które zostały wymienione w podrozdziale 1.3.1.

Siły Powietrzne Stanów Zjednoczonych, w świetle programu ochrony wykorzystywanego oprogramowania, definiują podatność na jako jego niedoskonałość, która pozwala atakującemu naruszyć bezpieczeństwo informacji. Według tej definicji podatność może być uznana jako podłoże do ataku na system, mające silny związek z trzema zasadniczych obszarami tj. [29]:

- występowanie podatności w systemie – funkcje lub zasoby systemu narażone na zagrożenia;
- dostęp cyberagresora do podatności w systemie – uzyskanie fizycznego lub zdalnego dostępu do podatności w systemie (np. dostęp fizyczny, przez sieć, przez Internet);

- zdolność wykorzystania podatności w systemie przez cyberagresora – możliwość pozyskania wiedzy oraz narzędzi niezbędnych do wykorzystania funkcji i/lub zasobów systemu poprzez jego podatności.

W kontekście funkcjonowania organizacji podatności definiuje się jako wady lub luki [18]:

- w strukturze fizycznej organizacji,
- sprzęcie i oprogramowaniu,
- zarządzaniu i administrowaniu,
- organizacji pracy,
- procedurach,
- obsadzie stanowisk personelem,

które mogą być wykorzystane przez zagrożenia do spowodowania szkód w systemie informacyjnym organizacji lub w jej działalności.

W niniejszej pracy przyjęto ogólną definicję podatności w odniesieniu do infrastruktury krytycznej [30]:

Def 1.1.6

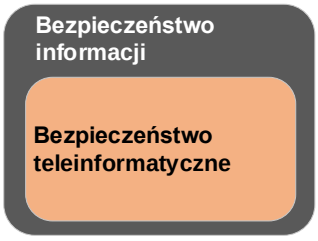
Podatność *to słabość infrastruktury występująca w jej komponentach – elementach technicznych, organizacyjnych, proceduralnych, która po wykorzystaniu powoduje negatywny wpływ na realizowalność usługi oraz jej dostępność.*

Łagodzenie podatności w tym kontekście może obejmować zmiany w konstrukcji, organizacji procesów realizacji usługi, zabezpieczeniach, kontroli stanu komponentów, zarówno co do częstości jak i metod kontrolowania komponentów infrastruktury, procedur formalno-prawnych, w tym ubezpieczenia ryzyka, ale może również obejmować zmiany w specyfikacji lub nawet deprecjację specyfikacji (np. usunięcie w całości podatnych na wpływ negatywny elementów technicznych lub programowych, czyli pewnych funkcjonalności, które mogą okazać się zbędne w procesie realizacji usług lub w niewielkim stopniu ich brak pogarsza jakość usługi).

W celu zaznaczenia różnicy pomiędzy podatnością, a zagrożeniem istotne jest, aby zaznaczyć, że na zagrożenia nie mamy wpływu, natomiast zagrożenie nie zawsze się musi zrealizować. Z drugiej strony, na podatności mamy wpływ, jako że możemy neutralizować lub minimalizować ich ilość, stosując zabezpieczenia i poprawki [18].

Podatności powinny być katalogowane i poddane szczegółowej analizie w kontekście zagrożeń wewnętrznych i zewnętrznych. Warto podkreślić, że w terminologii fachowej klasyfikowane są na wiele sposobów. Mogą one dotyczyć między innymi wykorzystywanego sprzętu, oprogramowania, architektury sieciowej jak i działalności samych użytkowników systemu. Jednakże istotne jest, aby zwrócić szczególną uwagę na to, że każda podatność może być wykorzystana przez potencjalnego atakującego. Podatności ściśle związane z oprogramowaniem będące przedmiotem zainteresowania niniejszej pracy można uznać jako podzbiór zbioru wszystkich podatności na zagrożenia w cyberprzestrzeni.

1.1.6 Bezpieczeństwo informacji, bezpieczeństwo teleinformatyczne, zapewnienie bezpieczeństwa teleinformatycznego



Rys. 7. Relacja pomiędzy bezpieczeństwem informacji a bezpieczeństwem teleinformatycznym

Definicja 1.1.7

Bezpieczeństwo informacji *jest to stopień uzasadnionego zaufania (poparty np. analizą ryzyka i przyjętymi metodami postępowania z ryzykiem) jednostki, grupy społecznej, całego społeczeństwa co do dostępności i jakości pozyskiwanej, przechowywanej, wykorzystywanej i przekazywanej informacji. Zakłada ono że nie zostaną poniesione potencjalne straty wynikające z jej ujawnienia, zniszczenia, modyfikacji czy uniemożliwienia przetwarzania.* [1]

Definicja 1.1.8

Bezpieczeństwo teleinformatyczne *jest to stopień uzasadnionego zaufania, że nie zostaną poniesione potencjalne straty wynikające z niepożądanego (przypadkowego lub świadomego) ujawnienia, modyfikacji, zniszczenia lub uniemożliwienia przetwarzania informacji przechowywanej, przetwarzanej i przesyłanej za pomocą systemów teleinformatycznych* [31].

W ujęciu formalnym bezpieczeństwo można określić jako uporządkowaną „czwórkę” [1]:

$$B =< Z,\ R,\ S,\ \Omega > \tag{1.2}$$

gdzie:

Z - jest niepustym zbiorem zagrożeń;

R - jest zbiorem podatności, które mogą zostać wykorzystane podczas realizacji zagrożenia ze zbioru Z;

S - jest zbiorem skutków (szkód) realizacji zagrożeń;

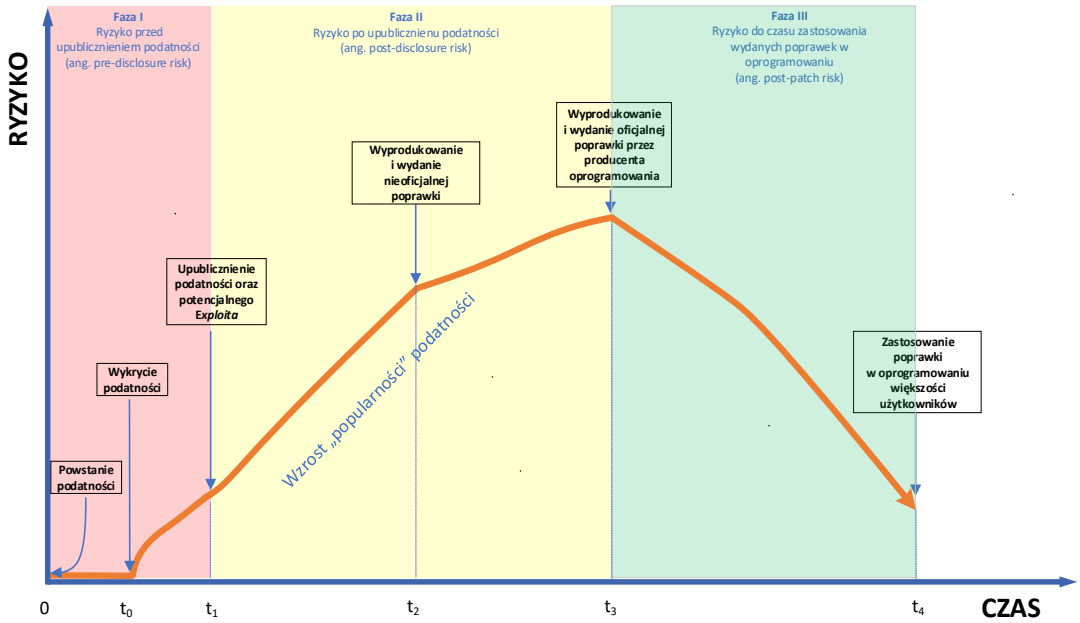
Ω - jest niepustym podzbiorem relacji określonych w iloczynie kartezjańskim $Z \times R \times S$ wiążących elementy zbioru Z z podatnościami ze zbioru R, których wykorzystanie przez zagrożenie powoduje skutki (szkody) ze zbioru S.

Definicja 1.1.9

Zapewnienie bezpieczeństwa teleinformatycznego *jest to zespół działań organizacyjnych i technicznych mających na celu minimalizację ryzyka utraty informacji i/lub zakłócenia działania systemów teleinformatycznych.*

1.2 Cykl życia podatności oprogramowania

Podobnie jak w procesie wytwarzania oprogramowania również w stosunku do podatności oprogramowania można mówić o ich cyklu życia (ang. *Vulnerability Life Cycle*). Cykl życia podatności, opierając się na dotychczasowych badaniach, określa etapy, w których w zależności od upływającego czasu zmienia się ryzyko wykorzystania podatności do szkodliwych działań. Można w nim wyodrębnić trzy zasadnicze fazy, które zobrazowano na Rys. 8:



Rys. 8. Cykl życia podatności. Źródło: Opracowanie własne na podstawie: [15] [32] [33]

W dalszej części przedstawiono skróconą interpretację poszczególnych faz cyklu życia podatności [15]:

- Faza I – ryzyko przed upublicznieniem podatności (ang. *pre-disclosure risk*), obejmuje poniższe zdarzenia $(0;\ t_1]$:
    - powstanie/narodziny podatności (0),
    - wykrycie podatności (t_0),
    - upublicznienie podatności (ang. *disclosure*) oraz potencjalnych możliwości jej wykorzystania – eksploita[1] (t_1).

Wykrycie istniejącej podatności w oprogramowaniu w chwili t_0 następuje w wyniku poszukiwań przez:

- kompetentne osoby lub grupy badawcze (akademickie, producentów oprogramowania, producentów oprogramowania antywirusowego, itp.) zajmujące się bezpieczeństwem oprogramowania;
- osoby zamierzające wykorzystać znalezione błędy do celów przestępczych.

Jeżeli dowiedzione zostanie, że podatność może posłużyć jako tak zwana furtka do przeprowadzenia ataku, np. za pomocą wykonanego prototypu oprogramowania – eksploita, a jednocześnie wiedza ta była posiadana przez człowieka, wówczas ryzyko szkodliwych działań rośnie, co można zaobserwować jako niewielki wzrost nachylenia krzywej na wykresie w przedziale $(t_0;\ t_1]$. Zwieńczeniem fazy I cyklu życia podatności, jest upublicznienie wiedzy na temat podatności oraz często także o sposobie jej wykorzystania – eksploicie (chwila t_1).

- Faza II – ryzyko po upublicznieniu podatności (ang. *post-disclosure risk*), obejmuje poniższe zdarzenia $(t_1;\ t_3]$:
    - Wyprodukowanie i wydanie nieoficjalnej poprawki (łatki) oprogramowania (t_2),
    - Wyprodukowanie i wydanie oficjalnej poprawki (łatki) oprogramowania (t_3),

Fragment krzywej z wykresu dla przedziału czasu $(t_1;\ t_2]$ charakteryzuje gwałtowny wzrost ryzyka wykorzystania podatności do szkodliwych działań. Spowodowane

1 Eksploit (ang. e*xploit*) – złośliwe aplikacje pozwalające na włamanie do komputera ofiary przy wykorzystaniu podatności w zainstalowanym oprogramowaniu [65].

jest to m. in. faktem, iż z upływem czasu wiedza o badanej podatności staje się coraz bardziej powszechna, przez co nastąpić może wzrost potencjalnej liczby osób zainteresowanych do wykorzystania jej jako podłoża ataku cybernetycznego. Kolejne etapy w fazie II wskazują na wyprodukowanie i wydanie poprawek oprogramowania pozwalających podatność zneutralizować, zarówno tych nieoficjalnych jak i wydanych oficjalnie przez producenta oprogramowania. Skutkuje to wyhamowaniem wzrostu wartości krzywej ryzyka wykorzystania podatności do szkodliwych działań.

- Faza III – ryzyko do czasu zastosowania wydanych poprawek w oprogramowaniu, obejmująca zdarzenie$(t_3;\ t_4]$:
    - Zastosowanie łatki w oprogramowaniu (t_4).

Od chwili t_3 maleje ryzyko wykorzystania podatności do szkodliwych działań, aż do chwili t_4, po której jeśli poprawka skutecznie wyeliminowała podatność, cykl życia podatności zostaje zakończony.

Należy jednak zaznaczyć, że w każdej z powyższych faz cyklu życia podatności może dojść do realizacji ataku cybernetycznego jeśli zostanie ona skutecznie wykorzystana podczas realizacji potencjalnego zagrożenia.

1.3 Taksonomie podatności

Znajomość podatności na zagrożenia w cyberprzestrzeni pozwala zrozumieć w jaki sposób podmioty atakujące mogą potencjalnie naruszyć bezpieczeństwo systemów w celu ich skompromitowania. Podatności posiadają zróżnicowane cechy, a z doświadczeń organizacji analizujących incydenty komputerowe wiadomo, iż pewne grupy podatności są łatwiejsze do wykorzystania przez atakującego, inne zaś mogą wymagać o wiele więcej wysiłku np. stosowanie technik inżynierii społecznej [34]. Warto zwrócić uwagę, iż zbieranie informacji o podatnościach wykrytych w cyberprzestrzeni jest procesem pozbawionym spójności. Spowodowane jest to występowaniem różnorodnych źródeł informacji o podatności, gdzie część z nich może nie być udostępniona do powszechnego użycia. Faktem zatem jest, iż wskazanie liczby wszystkich wykrytych podatności występujących w cyberprzestrzeni do danej chwili nie jest możliwe. Ponadto, niekompletność informacji o podatnościach oraz przeprowadzane aktualizacje badanego oprogramowania, przyczyniają się do błędów w statystykach podatności oraz w interpretacjach w odniesieniu do stwarzanego przez nie zagrożenia.

Istniejące przykłady klasyfikacji podatności tworzone były zatem w oparciu o różne atrybuty podatności. Zestawienie klasyfikacji podatności zebranych na etapie przeprowadzonej analizy literatury przedstawiono w Tabela 1.

Tabela 1. Klasyfikacje podatności. Źródło: Opracowanie własne.

Nazwa/autor, rok publikacji	Atrybuty klasyfikacji	Cel	Dodatkowe informacje
McPhee, 1974 [35]	Cechy podatności wynikające z błędnych założeń projektowych	Identyfikacja podatności systemów operacyjnych	Próba wyodrębnienia indywidualnych cech charakterystycznych podatności
RISOS Project, 1976 [36]	Ustalone cechy podatności lub wyniki działania podatnych elementów systemu operacyjnego	Identyfikacja i ocechowanie podatności systemów operacyjnych	-
Aslam, 1995 [37]	Luki bezpieczeństwa w systemach UNIX	Wyodrębnienie i uporządkowanie informacji o podatnościach w celu utworzenia dedykowanej bazy danych podatności	-
Bishop, 1999 [38]	Parametry podatności zebrane przez systemy IDS	Uporządkowanie oraz ustalenie formatu opisu podatności systemów UNIX oraz podatności protokołów sieciowych	Wynikowy zbiór podatności miał być wykorzystany do automatyzacji działania systemów IDS
VERDICT, 2001 [39]	Ustalone subiektywnie cechy podatności tj. m.in. losowość, warunki wystąpienia, walidacja, poziom zagrożenia	Identyfikacja i ocechowanie podatności systemów operacyjnych	Charakterystyka podatności oraz wskazanie potencjalnego mechanizmu zapobiegawczego (obronnego)
Piessens, 2002 [40]	Podatności oprogramowania	Identyfikacja i ocechowanie najczęściej występujących podatności oprogramowania	Podatności dla poszczególnych faz cyklu życia oprogramowania
Andy Gray, 2003 [41]	-	Próba wykorzystania i ujednolicenia istniejących taksonomii podatności	Wykorzystanie wcześniej utworzonych taksonomii podatności
Pothemsetty and Akyol, 2004 [42]	Podatności protokołów sieciowych	Próba kategoryzacji podatności protokołów sieciowych	Ekstrakcja wybranych cech lub sposobów działania protokołów sieciowych
Seacord, 2005 [43]	Podatności oprogramowania	Dostarczyć wielowymiarowy opis cech charakterystycznych podatności	Wynikowy zbiór podatności miał być wykorzystany do automatyzacji procesu analizy podatności
Fortify Taxonomy, 2006 [44]	Ustalone klasy błędów popełnianych podczas projektowania oprogramowania	Zbadać i zrozumieć podstawowy zbiór typowych błędów skutkujących podatnościami, podczas tworzenia oprogramowania	Rozpatrywano m.in. walidację wprowadzanych danych, zabezpieczenia API, jakość kodu, enkapsulację, środowisko docelowe
Bazaz and Arthur, 2007 [45]	Podatności systemów operacyjnych i oprogramowania	Ustalenie relacji pomiędzy zasobami, procesami oraz podatnościami w systemach operacyjnych i oprogramowania	Bazowano na źródle oraz zasobach badanego systemu operacyjnego.
IGURE, 2008 [46]	Podatności oprogramowania	Próba kategoryzacji podatności w oparciu o wnioski z cyberataków	Uwzględniono m.in. wektor cyberataku, wpływ na działanie zaatakowanego

			systemu, charakterystykę zaatakowanego systemu
ISO 27005:2008 [24]	Sprzęt komputerowy, oprogramowanie, sieć, obsługa, teren, organizacyjne	Podział podatności na obszary	Opisano w 1.3.1
NIST/MITRE [34] [47] [48] [49]	Podatności oprogramowania (CVE), podatności konfiguracji (CCE), podatności wynikające z niepoprawnego użytkowania oprogramowania	Wyodrębnienie obszarów podatności w celu utworzenia dedykowanych baz danych podatności.	Opisano w 1.3.2

W kolejnych podrozdziałach przedstawiono charakterystykę dwóch wybranych taksonomii podatności.

1.3.1 Klasyfikacja podatności według normy ISO 27005:2008

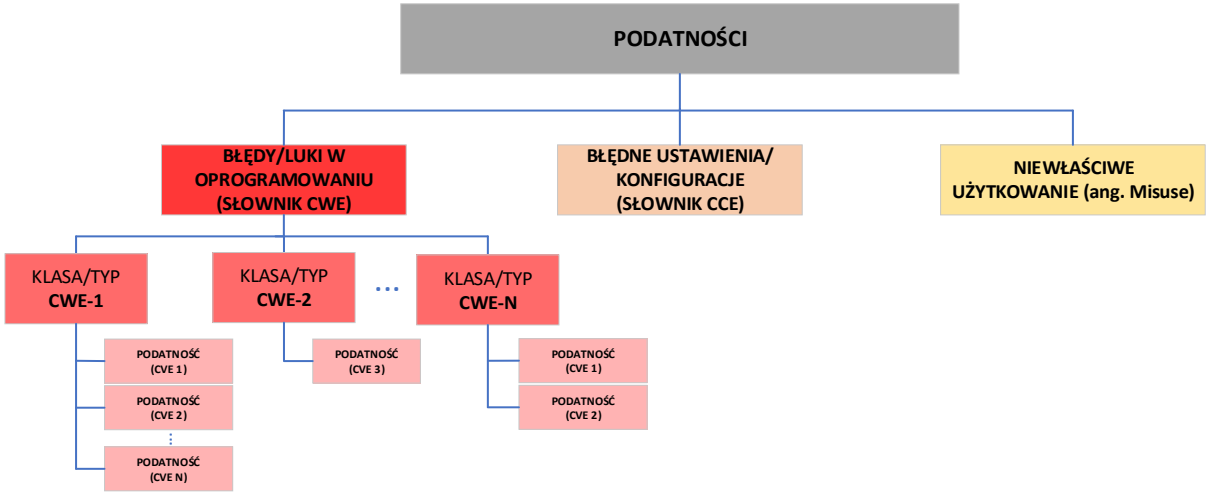
Według normy ISO 27005:2008 podatności sklasyfikowane zostały na obszary, takie jak [28]:

- sprzęt komputerowy (ang. *hardware*), np.:
    - wrażliwość na wilgoć
    - wrażliwość na kurz
    - podatność na zabrudzenia
    - podatność na niezabezpieczone przechowywanie
- oprogramowanie (ang. *software*), np.:
    - brak testów
    - nieodpowiednie testowanie
- sieć (ang. *network*), np.:
    - niezabezpieczone łącza
    - architektura sieci podatna na ataki
- obsługa (ang. *personel*), np.:
    - nieodpowiedni proces rekrutacji pracowników
    - niewystarczające poświadczenie bezpieczeństwa
- teren (ang. *site*), np.:
    - zagrożony klęskami żywiołowymi
    - niestabilne źródło zasilania
- organizacyjne (ang. o*rganizational*), np.:

- brak regularnych testów
- brak planów związanych z bezpieczeństwem
- brak planów ciągłości działania organizacji

1.3.2 Klasyfikacja podatności - wyniki prac kierowanych przez NIST

Jednym z podstawowych problemów w procesie projektowania bezpieczeństwa systemów, w szczególności na poziomie technicznym, jest przedstawienie podatności w jednolity, znany i powszechnie akceptowany sposób. Taki sposób opisu dostarczony został w wyniku realizacji prac kierowanych przez amerykański instytut NIST[2] i można go przedstawić następująco:



Rys. 9. Schemat podziały podatności systemów informatycznych na podstawie prac kierowanych przez NIST. Źródło: Opracowanie własne na podstawie [34] [47] [48] [49]

Pozwala on wyodrębnić następujące klasy podatności: [34] [47] [48] [49]:

- *luki w oprogramowaniu*[3] – spowodowane błędem w projektowaniu lub implementacji oprogramowania, zarówno niezamierzonym jak i zamierzonym;
- *błędne konfiguracje systemu/oprogramowania*[4] – niedoskonałości konfiguracji systemu, poprzez które atakujący może uzyskać do niego dostęp;
- *podatności związane z niepoprawnym użytkowaniem systemu/oprogramowania* – niewłaściwe wykorzystanie funkcji oprogramowania, prowadzące do naruszenia

[2] NIST (*National Institute of Standards and Technology*) - amerykańska agencja federalna spełniająca funkcję analogiczną do *Głównego Urzędu Miar*
[3] Na potrzeby katalogowania podatności z tej grupy utworzono słowniki *Common Vulnerabilities and Exposures* (*CVE®*) oraz *Common Weakness Enumeration* (*CWE*);
[4] Na potrzeby katalogowania podatności z tej grupy utworzono słownik *Common Configuration Enumeration* (*CCE™*)

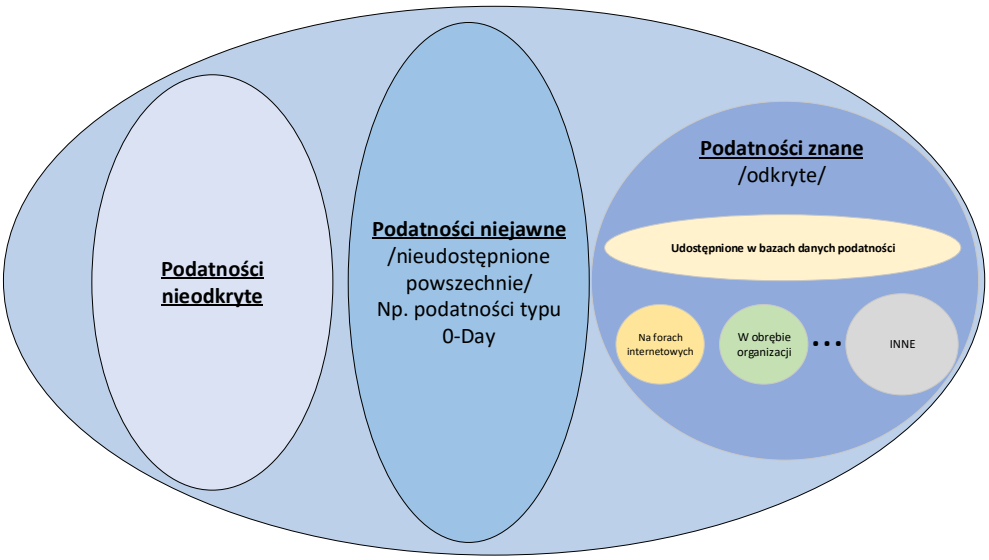
bezpieczeństwa systemu, np. odebranie za pomocą komunikatora zainfekowanego pliku przesłanego przez atakującego.

W myśl autorów tego podejścia stosowanie jednolitej konwencji nazewniczej, interfejsów czy formatów danych może zautomatyzować proces wymiany danych o podatnościach pomiędzy systemami lub ich komponentami, co może skutkować wzrostem poziomu bezpieczeństwa, a jednocześnie wzrostem poziomu interoperacyjności dla projektowanych systemów informatycznych.

1.4 Analiza ilościowa podatności

1.4.1 Źródła danych podatności

Proces analizy podatności w ujęciu statystycznym posiada pewne ograniczenia, które wynikają przede wszystkim ze specyfiki danych źródłowych, gdzie jednocześnie część z nich może nie być udostępniona do powszechnego użycia. Należy zatem zwrócić uwagę na fakt, iż nie jest możliwe zebranie pełnej informacji o wszystkich rodzajach podatności, które mogą wystąpić w oprogramowaniu, a za przykład mogą tutaj posłużyć podatności typu *0-day*, o których wiedzę posiadają jedynie wybrane organizacje lub grupy użytkowników. Ponadto, niekompletność informacji o podatnościach oraz przeprowadzane aktualizacje badanego oprogramowania, przyczyniają się do błędów w statystykach podatności oraz w interpretacjach w odniesieniu do stwarzanego przez nie zagrożenia (ang. *vulnerability severity*).



Rys. 10. Poglądowy schemat przedstawiający podział zbioru podatności. Źródło: Opracowanie własne na podstawie: [50].

Niemniej jednak, aby usprawnić pracę z podatnościami, a jednocześnie aby statystyki podatności przedstawiano w sposób jasny i ujednolicony podjęto próbę utworzenia:

- baz danych podatności (ang. *vulnerability databases*),
- biuletynów bezpieczeństwa (ang. *security bulletins*).

Przyjmuje się, że w bazach danych podatności zazwyczaj jednym identyfikatorem oznaczana jest jedna podatność, natomiast w biuletynach bezpieczeństwa zazwyczaj wykazywane są podzbiory podatności danego oprogramowania, dla którego przygotowano poprawki bezpieczeństwa (ang. *patches*). Zestawienie baz danych podatności oraz biuletynów bezpieczeństwa zaprezentowano w Tabela 2.

Tabela 2. Źródła danych podatności. Opracowanie własne

Nazwa źródła danych	Rodzaj udostępnianego źródła danych	Informacje ogólne	Przykład
Apache	Biuletyny bezpieczeństwa	Biuletyn bezpieczeństwa dla produktu: *Apache HTTP Server*	CVE-2017-9798
BDU Fstec	Połączenie baz danych podatności i biuletynów bezpieczeństwa - źródło rządowe	Baza danych zagrożeń bezpieczeństwa informacji – rosyjska, rządowa	BDU:2018-00749
Beyond Security SecuriTeam	Baza danych pojedynczych podatności - komercyjne skanery i narzędzia agregacji danych podatności	Baza danych podatności zarządzana przez organizację, której celem jest usprawnienie procesu zarządzania podatnościami (ang. *Vulnerability Management vendor*)	CVE-2016-9939
CentOS CESA	Biuletyny bezpieczeństwa - repozytorium dla konkretnej wersji oprogramowania	Repozytorium – biuletyny bezpieczeństwa dla systemu operacyjnego CentOS (*The CentOS-announce Archives*)	CESA-2014:0376
CIS OVAL	Połączenie baz danych podatności i biuletynów bezpieczeństwa - publiczne, sformalizowane reguły detekcji podatności	Repozytorium definicji reguł detekcji podatności *OVAL* (*Open Vulnerability and Assessment Language*) utworzone i zarządzane przez głównego ich producenta	b/d
CISCO SA	Biuletyny bezpieczeństwa	Biuletyny bezpieczeństwa organizacji CISCO (*Cisco Security Advisories*)	cisco-sa-20180521
CNNVD	Baza danych pojedynczych podatności - źródło rządowe	Chińska baza danych podatności (*China National Vulnerability Database of Information Security*)	CNNVD-201803-1136
CNVD	Baza danych pojedynczych podatności - źródło rządowe	Chińska baza danych podatności (*China National Vulnerability Database*)	CNTA-2018-0012
Debian DSA	Biuletyny bezpieczeństwa - repozytorium dla konkretnej wersji oprogramowania	Repozytorium – biuletyny bezpieczeństwa dla systemu operacyjnego Debian (*Debian Security Advisories*)	DSA-4156-1
Flexera	Baza danych pojedynczych podatności - komercyjne skanery i narzędzia agregacji danych podatności	Repozytorium danych zarządzane przez organizację, której celem jest usprawnienie procesu odkrywania podatności (ang. *Vulnerability Intelligence vendor*)	b/d
IBM X force	Baza danych pojedynczych podatności	Baza danych podatności udostępniana przez zespół zajmujący się badaniami w dziedzinie bezpieczeństwa informatycznego	CVE-2018-7600
JVN	Połączenie baz danych podatności i biuletynów bezpieczeństwa - źródło rządowe	Japońskie repozytorium danych o podatnościach (*Japan Vulnerability Notes*)	JVN#65268217
Microsoft KB	Biuletyny bezpieczeństwa	Baza wiedzy zawierająca artykuły redagowane wykwalifikowanych pracowników Pomocy technicznej Microsoft (*Microsoft Knowledge Base*)	KB4012289
Microsoft MS	Biuletyny bezpieczeństwa	Biuletyn bezpieczeństwa firmy Microsoft (*Microsoft Security Bulletin*)	MS17-10
Mitre CVE	Baza danych pojedynczych podatności	Baza danych podatności utworzona przez główną organizacja koordynująca rejestrację podatności pod identyfikatorem CVE, uznanym za standard wymiany danych podatności. Zawiera dane służące	CVE-2018-7600

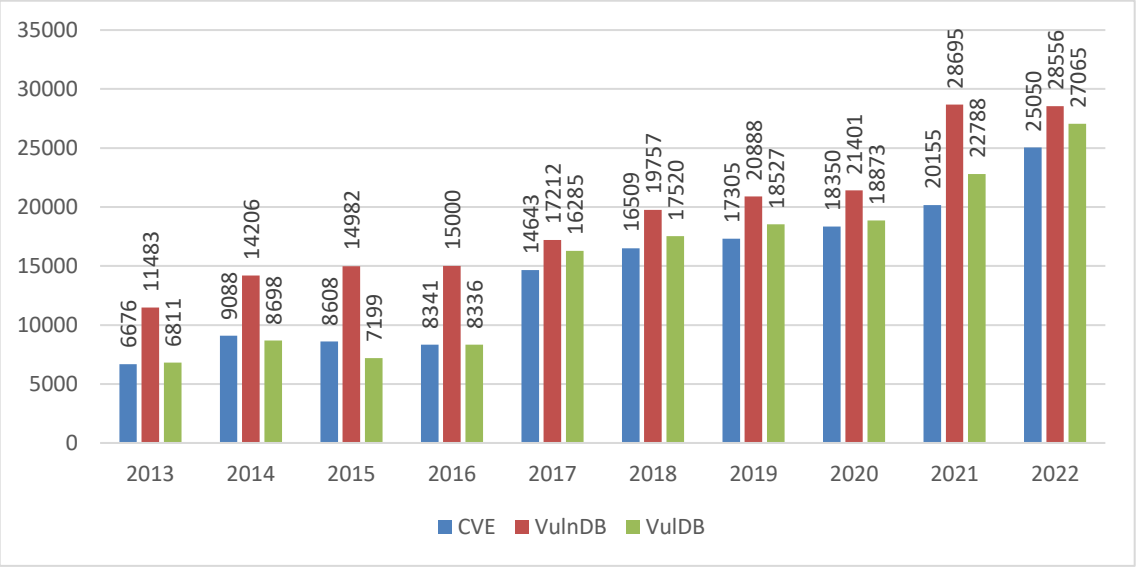
		do zasilenia wielu innych baz danych podatności (np. NVD)	
Mozilla MFSA	Biuletyn bezpieczeństwa	Biuletyny bezpieczeństwa, w których informuje o występowaniu luk w zabezpieczeniach programów Mozilla Firefox oraz Mozilla Thunderbird Mozilla Foundation Security Advisories	mfsa2018-13
NIST NVD	Baza danych pojedynczych podatności - źródło rządowe	Amerykańska (rządowa) baza danych podatności - utworzona w rezultacie badań przewodzonych przez amerykański instytut NIST. Baza danych zasilana jest danymi z repozytorium MITRE CVE.	CVE-2018-7600
OpenSSL Vulnerabilities	Biuletyny bezpieczeństwa	Repozytorium danych podatności występujących w wieloplatformowej, otwartej implementacji protokołów SSL i TLS oraz algorytmów kryptograficznych ogólnego przeznaczenia.	CVE-2018-0737
Rapid7	Połączenie baz danych podatności i biuletynów bezpieczeństwa - komercyjne skanery i narzędzia agregacji danych podatności	Komercyjne repozytorium podatności oraz eksploit'ów (*Vulnerability & Exploit Database*)	CVE-2018-7600
RedHat RHSA	Biuletyny bezpieczeństwa - repozytorium dla konkretnej wersji oprogramowania	Repozytorium – biuletyny bezpieczeństwa dla systemu operacyjnego RedHat (*The RedHat-announce Archives*)	RHSA-2014:0376
Risk Based Security VulnDB	Baza danych pojedynczych podatności - komercyjne skanery i narzędzia agregacji danych podatności	Repozytorium danych zarządzane przez organizację, której celem jest usprawnienie procesu odkrywania podatności (ang. *Vulnerability Intelligence vendor*)	b/d
SecPod SCAP Repo	Połączenie baz danych podatności i biuletynów bezpieczeństwa - komercyjne skanery i narzędzia agregacji danych podatności	Repozytorium definicji reguł detekcji podatności *OVAL* (*Open Vulnerability and Assessment Language*) utworzone i zarządzane przez organizację *SecPod*	CVE-2018-7600
Tenable Nessus plugins	Połączenie baz danych podatności i biuletynów bezpieczeństwa - komercyjne skanery i narzędzia agregacji danych podatności	Repozytorium danych oraz testów do wykrywania podatności, tworzone na potrzeby komercyjnego skanera podatności	CVE-2018-7600
Ubuntu USN	Biuletyny bezpieczeństwa - repozytorium dla konkretnej wersji oprogramowania	Repozytorium – biuletyny bezpieczeństwa dla systemu operacyjnego Ubuntu (*Ubuntu security notices*)	USN-2165-1
VulDB	Baza danych pojedynczych podatności - komercyjne skanery i narzędzia agregacji danych podatności	Repozytorium danych oraz skaner podatności zarządzane przez organizację, której celem jest usprawnienie procesu odkrywania podatności (ang. *Vulnerability Intelligence vendor*)	CVE-2018-7600
Vulners	Połączenie baz danych podatności i biuletynów bezpieczeństwa - komercyjne skanery i narzędzia agregacji danych podatności	Repozytorium danych oraz skaner podatności zarządzane przez zespół inżynierów, których celem jest usprawnienie procesu odkrywania i zarządzania podatnościami	CVE-2018-7600

Zarówno bazy danych podatności, jak i biuletyny bezpieczeństwa udostępniają dane obejmujące wiele parametrów podatności, charakteryzujące m.in. ich pochodzenie, rodzaj czy stopień stwarzanego zagrożenia, co może stanowić istotną informację z punktu widzenia projektowania bezpiecznego oprogramowania. Spośród wymienionych przykładów źródeł danych podatności wiele z nich bazuje na otwarto-źródłowym repozytorium podatności CVE MITRE, w którym podatności ujęte są za pomocą ustandaryzowanego formatu opisu. W zestawieniu zaprezentowano przykłady znanych repozytoriów danych podatności udostępniane komercyjnie oraz publicznie, z uwzględnieniem kilku repozytoriów oficjalnie wykorzystywanych przez organizacje rządowe. Koncepcja gromadzenia informacji w przypadkach rozwiązań komercyjnych jest podobna jak dla wielu darmowych repozytoriów, jednakże, jak zaznaczają autorzy, zrealizowana jest w bardziej kompleksowy

sposób, a informacje o podatnościach mogą obejmować więcej szczegółów. Dostępne są w nich także dodatkowe narzędzia, między innymi do analizy, agregacji lub przeszukiwania zbioru podatności, które w znacznym stopniu mogą ułatwić współpracę z samym repozytorium. Niemniej jednak uzyskanie dostępu do źródeł komercyjnych podatności w wymaga najczęściej wykupienia subskrypcji.

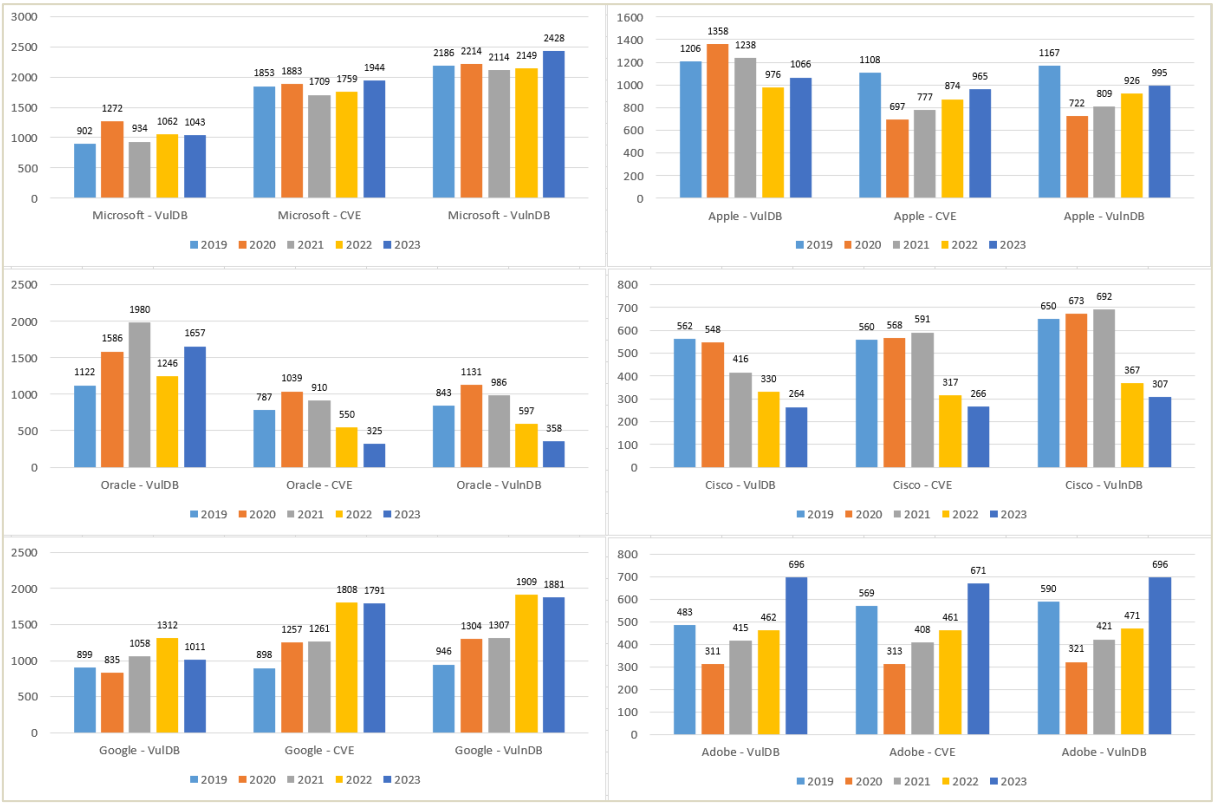
1.4.2 Podatności w ujęciu liczbowym

Zestawienie liczby podatności przedstawiono na wykresie na Rys. 11. Zestawienie to uwzględnia wszystkie podatności, które zarejestrowano w bazach danych podatności *NVD*, *VulnDB* oraz *VulDB* w latach 2013-2022.



Rys. 11. Statystyka przedstawiająca liczbę podatności zarejestrowanych w bazach danych NVD (wpisy CVE), VulnDB oraz VulDB w latach 2013-2022. * Stan na dzień 23 stycznia 2024. Źródło: opracowanie własne na podstawie [51] [52] [53]

Na podstawie analizy podatności w ujęciu liczbowym w latach 2013-2022 zaprezentowanych na Rys. 11 można zaobserwować, iż w wybranych trzech bazach danych podatności zarejestrowanych zostało łącznie od kilku nawet do kilkudziesięciu tysięcy podatności rocznie. Dla analizowanego okresu czasu, jednoznacznie można stwierdzić, że komercyjna baza danych podatności VulnDB w każdym kolejnym roku zawierała więcej rekordów wykrytych podatności niż w przypadku pozostałych. Pomijając próbę porównania skuteczności w detekcji podmiotów zasilających ją danymi podatności różnica ta może być również zwiększona m.in. tym, iż autorzy rozwiązania VulnDB poza oprogramowaniem flagowych producentów na rynku światowym, wyszukują podatności dodatkowo w ponad 2000 bibliotekach mniej znanych producentów oprogramowania (ang. *3rd-party libraries*).



Rys. 12. Statystyka przedstawiająca porównanie liczb podatności dla wybranych sześciu znanych producentów oprogramowania w latach 2019-2023 pomiędzy bazą danych NVD, VulnDB oraz VulDB. * Stan na dzień 12 kwietnia 2024. Źródło: Opracowanie własne na podstawie [51] [52] [53]

Na podstawie opracowanego zestawienia na wykresie na Rys. 12 można jednoznacznie stwierdzić, że w wyniku analiz przeprowadzonych dla oprogramowania flagowych organizacji takich jak Microsoft, Apple, Google, Cisco, Adobe i Oracle w latach 2019-2023 wykryto i zarejestrowano w tych trzech niezależnych zbiorach danych tysiące podatności, co potwierdza, że podatności mogą być poważnym problemem dla ich użytkowników. Zaprezentowane statystyki wskazują również, iż pomimo postępującego rozwoju technologicznego w kontekście zwalczania podatności, nie zauważa się spadku ich ogólnej liczebności. Jednocześnie, wraz ze wzrostem liczby użytkowników cyberprzestrzeni, w przypadku korzystania przez nich z podatnego oprogramowania, może nastąpić wzrost liczby potencjalnych celów ataku cybernetycznego.

Zarejestrowane podatności oprogramowania można również próbować uszeregować w zależności od potencjalnych następstw, do których może doprowadzić ich wykorzystanie (za przykład następstwa w tym przypadku możemy uznać np. rodzaj przeprowadzonego ataku). Przykład takiego zestawienia na lata 2015-2023, z wyodrębnieniem sześciu wybranych kategorii zaprezentowano w Tabela 3.

Tabela 3. Statystyka podatności w latach 2015-2023 z podziałem na wybrany typ podatności. Źródło danych: Baza NVD. Opracowanie własne na podstawie [51]

Rok	Wykorzystanie podatności może prowadzić do:						
	Memory Corruption	**Overflow**	**Sql Injection**	**XSS**	**XXE**	**SSRF**	**CSRF**
2015	1104	1073	221	776	50	8	249
2016	1174	1214	97	497	41	16	88
2017	1555	2494	505	1500	109	57	334
2018	1748	2100	504	2043	189	118	479
2019	2-57	1213	554	2389	139	103	560
2020	1903	1222	466	2203	119	132	416
2021	2566	1677	744	2726	126	197	520
2022	3420	1886	1890	3407	127	235	769
2023	2812	1763	2159	5179	138	248	1398

1.5 Wnioski

Dokonując przeglądu literatury zaobserwowano, że w opracowaniach naukowych z dziedziny cyberbezpieczeństwa występują różnice w definicjach podstawowych pojęć. Przykładem nieprawidłowości terminologicznych może być wymienne i przez to błędne używanie pojęć np. zagrożenia i ataku czy zagrożenia i podatności. Jednocześnie zauważono, że utworzenie dobrze zdefiniowanej taksonomii podatności oprogramowania może usprawnić automatyzację procesów związanych z analizą kodu źródłowego podatności, a jednocześnie ujednolicić formę przekazu informacji o podatnościach pomiędzy podmiotami odpowiedzialnymi za projektowanie bezpieczeństwa oprogramowania. Stosowanie ustalonej klasyfikacji podatności może mieć również wpływ na realizację procesu pozyskiwania informacji o częstotliwości oraz trendach występowania określonych podzbiorów podatności w danym systemie, jak również związkach podatności z konkretnymi incydentami w cyberprzestrzeni.

Z przeprowadzonej analizy podatności w ujęciu liczbowym w latach 2013-2023 wynika, iż corocznie zidentyfikowanych zostaje tysiące nowych podatności. Ponadto, analizując dane z oficjalnych baz danych podatności można zaobserwować, że podatności występują w zasadzie w każdej instancji oprogramowania i jednocześnie posiadają zróżnicowane cechy, przez co występują różnice w poziomie stwarzanego przez nie zagrożenia, co może stanowić istotną informację z punktu widzenia projektowania bezpieczeństwa oprogramowania.

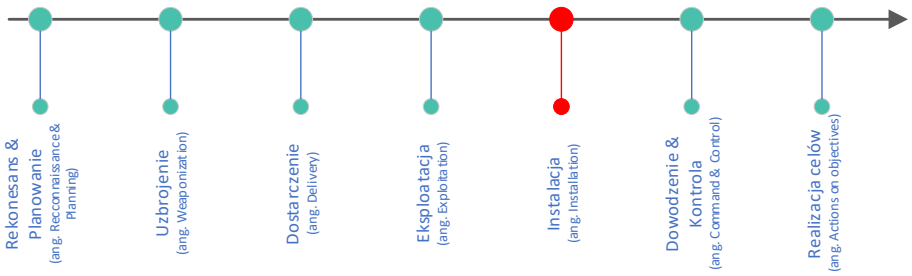
Obecnie, podmioty zajmujące się zapewnieniem bezpieczeństwa w cyberprzestrzeni prowadzą badania, których celem jest opracowanie metod oraz dedykowanych narzędzi usprawniających przeciwdziałanie, wykrywanie i neutralizację skutków cyberzagrożeń. Tego typu zagrożenia, będące w ogólnym ujęciu zdarzeniami w cyberprzestrzeni, mogą

powodować szkody w systemach zarówno użytkowników indywidualnych, jak i organizacji. Skuteczna walka z cyberzagrożeniami wymaga zatem wielkiego zaangażowania nie tylko specjalistów IT, ale również każdego jej użytkownika. Ponadto, źródła zagrożeń w cyberprzestrzeni podlegają ewolucji i mogą być zróżnicowane w zależności od pochodzenia i celu atakującego, który posiadając odpowiednią wiedzę o podatnościach po stronie potencjalnej ofiary, może przeprowadzić cyberatak o katastrofalnych skutkach. Zdobycie zatem informacji o podatnościach, daje zatem możliwość potencjalnemu atakującemu utworzenie metod i narzędzi, które mogą być wykorzystane do przeprowadzenia ataku. Z punktu widzenia bezpieczeństwa informacyjnego, istotne jest zatem posiadanie wiedzy o podatnościach lub wyodrębnionych cechach charakterystycznych dla podatności oraz posiadanie umiejętności przeciwdziałania podatnościom za pomocą metod technicznych. Jednocześnie cennym źródłem informacji o potencjalnych podatnościach jest prognozowanie ich wystąpienia za pomocą miarodajnych metod modelowania matematycznego do kompleksowej analizy lub prognozowania podatności, które w sposób skuteczny znajdują zastosowanie m. in. w badaniu niezawodności oprogramowania. Przykłady wybranych modeli i metod analizy podatności dostępnych w literaturze z obszaru badawczego przedstawiono w kolejnym rozdziale.

2. Przegląd modeli i metod analizy i prognozowania podatności występujących w literaturze

2.1 Wstęp

Dokonując analizy dotychczasowych strategii cyberataków można zaobserwować, iż za ich fundament można uznać pomyślne wykorzystanie przez atakującego wykrytej na etapie rekonesansu podatności, m.in. podatności oprogramowania, podatności w konfiguracji sprzętowej czy podatności wynikającej z błędnego korzystania przez użytkownika z oprogramowania. Przykładem może być scenariusz cyberprzestępczych kampanii typu APT (ang. *Advanced Persistent Threat*), uważanych według badań Kaspersky Lab za jedno z najpoważniejszych zagrożeń 2016 roku [54]. W scenariuszu ataku typu APT wyszczególniono siedem zasadniczych faz [55] (Rys. 13), gdzie w ujęciu ogólnym atakujący w pierwszej fazie zbiera informacje o potencjalnych podatnościach w systemie ofiary, a następnie w oparciu o rozpoznane podatności przygotowuje i dostarcza narzędzia ataku, po czym go wykonuje w celu do przejęcia kontroli i penetracji systemu ofiary.

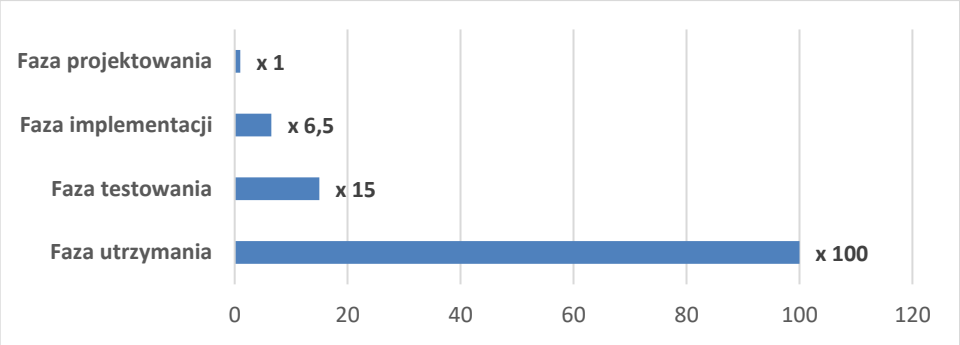


Rys. 13. Schemat przedstawiający fazy ataku APT na systemy komputerowe - *Kill Chain*. *Źródło: Opracowanie własne na podstawie* [55]

Wiedza atakującego dotycząca występowania podatności w oprogramowaniu potencjalnej ofiary może zatem stanowić poważny i wymierny problem dla jego użytkowników. Z drugiej strony świadomość posiadania podatności w oprogramowaniu oraz znajomość ich specyfikacji technicznej pozwalają zrozumieć w jaki sposób podmioty atakujące mogą zagrozić bezpieczeństwu informacji i w jaki sposób można ewentualnie zapobiec lub przeciwdziałać potencjalnemu atakowi.

W ostatnich latach podejmowanych zostało wiele prac nad zapewnieniem bezpieczeństwa oprogramowania, z uwzględnieniem występowania podatności na cyberzagrożenia. Ich głównym celem było najczęściej udoskonalanie technik i narzędzi wykrywania podatności oraz zapobiegania ich wystąpienia w oprogramowaniu, bazując na

wiedzy, będącej wynikiem wieloletnich badań i praktycznych doświadczeń instytucji lub firm zajmujących się cyberbezpieczeństwem. O wiele mniej miejsca w literaturze poświęcono na wykorzystanie metod ilościowych do badania podatności, które w sposób skuteczny i miarodajny znajdują zastosowanie np. w badaniu niezawodności oprogramowania [56]. Ponadto, podejmując próbę klasyfikacji podatności jako podzbioru wszystkich potencjalnych błędów oprogramowania, można zaobserwować, że ograniczenie ich liczby, szczególnie w początkowych fazach jego powstawania, w relatywnie znaczący sposób może obniżyć koszty jego utrzymania. Przykład rezultatów badań prezentujący tą zależność przedstawiono na Rys. 14.



Rys. 14. Wykres przedstawiający koszty naprawy błędów oprogramowania w poszczególnych fazach jego cyklu życia. Źródło: [57]

Obecnie w literaturze wyróżnić można wiele klasyfikacji podatności. W wyniku przeprowadzonej analizy literatury, w niniejszym podrozdziale przedstawiono i scharakteryzowano metody ilościowe analizy podatności, z podziałem na następujące grupy:

- mechanizmy i skale porównawcze (tzw. *systemy scoringowe*);

- modele wykrywania podatności;

- modele predykcji podatności.

2.2 Modele i metody ilościowe analizy podatności

Ilościowy proces analizy podatności może skupiać się zarówno na badaniu pojedynczej podatności, jak również na badaniu ich wyszczególnionych zbiorów, przykładowo obejmujących całe oprogramowanie lub jego moduły. Do tego celu utworzono metody, które mogą wskazać kierunek rozwoju mechanizmów zapewniających bezpieczeństwo oprogramowania. W przypadku analizy pojedynczej podatności proces ten może być zrealizowany w oparciu o mechanizmy i skale porównawcze, wykorzystujące

wyodrębnione charakterystyki podatności, wskazując ostatecznie np. jej stopień zagrożenia (ang. *vulnerability severity*) [34]. Skupiając się natomiast na badaniu zbiorów podatności docelowego oprogramowania, opracowano modele wykrywania podatności (ang. *Vulnerability Discovery Model- VDM*) oraz modele predykcji podatności (ang. *Vulnerability Prediction Model- VPM*).

2.2.1 Systemy scoringowe

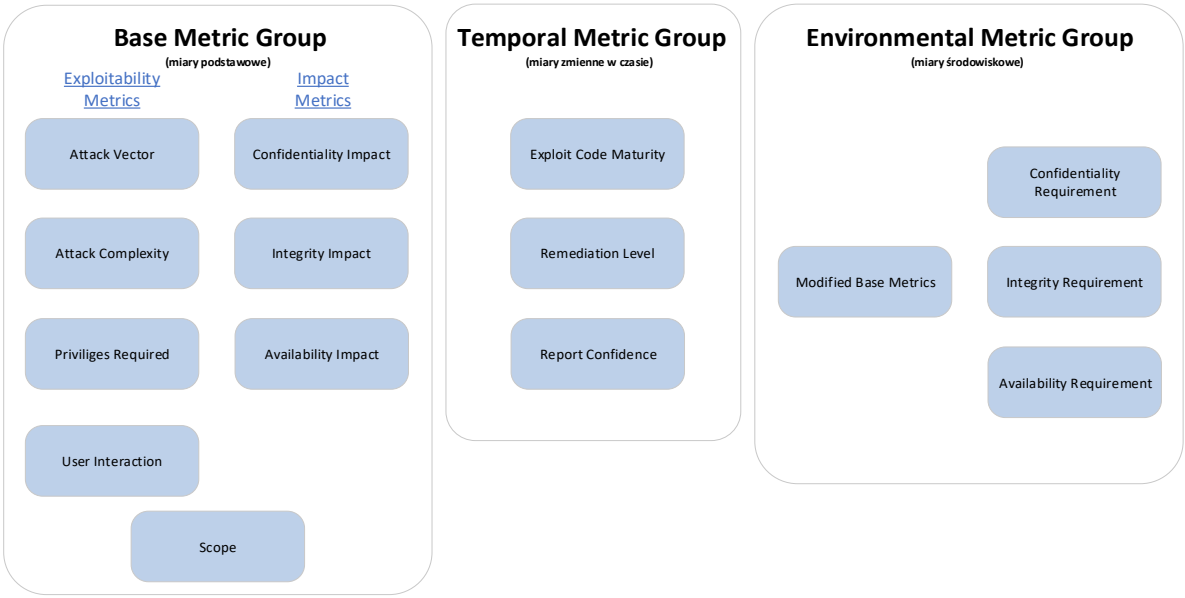
Systemy „scoringowe" działające na zasadzie skali i mechanizmów porównawczych umożliwiają pomiar stopnia zagrożenia pojedynczych podatności oprogramowania [34]. Analizowane podatności wyrażone są za pomocą wektora, którego składowe reprezentują jej odpowiednie charakterystyki – nazwane przez twórców miarami (ang. *Metrics*) [58] [59].

2.2.1.1 Systemy CVSS, CWSS, CCSS, CMSS

W literaturze najczęściej spotkać można czterech przykłady takich podejść, będących rezultatem badań prowadzonych pod kierownictwem amerykańskiego instytutu NIST (*National Institute of Standards and Technology*) t. j.:

- **CVSS (*Common Vulnerability Scoring System*)**

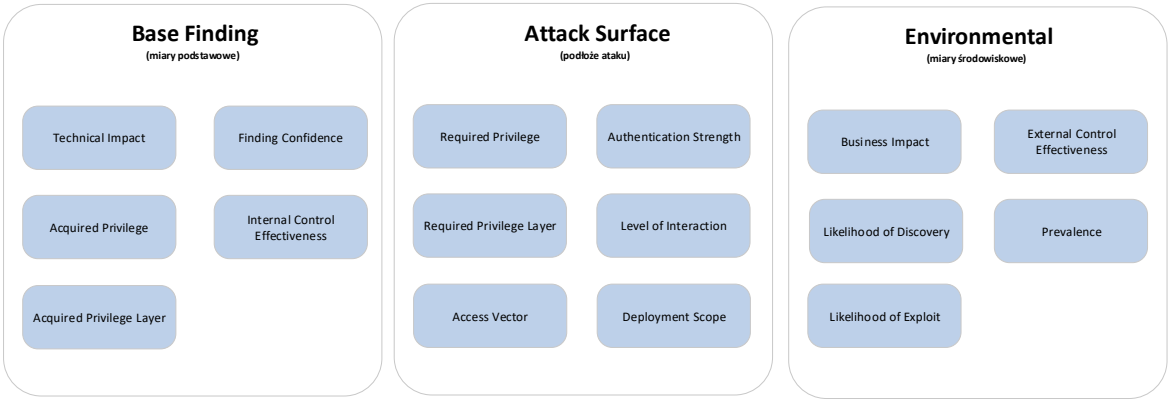
Mechanizm przeznaczony do badania stopnia zagrożenia podatności ze zbioru CVE (ang. *Common Vulnerabilities and Exposures*), wynikających z błędów w projektowaniu oraz tworzeniu oprogramowania oraz do wymiany informacji o charakterystykach podatności [34].



Rys. 15. Miary podatności zastosowane w standardzie CVSS. Źródło: [34]

- **CWSS (*Common Weakness Scoring System*)**

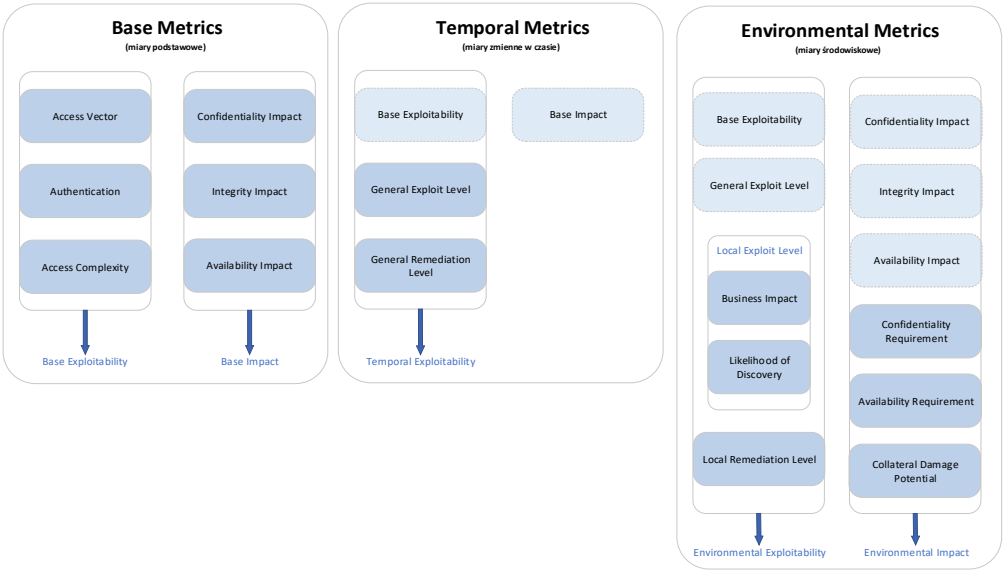
Mechanizm służący do badania stopnia zagrożenia podatności będących wynikiem wad w oprogramowaniu, sklasyfikowanych w zbiorze CWE (ang. *Common Weakness Enumeration*). Koncepcyjnie zbliżony jest do CVSS, natomiast występują różnice w budowie oraz rodzaju przetwarzanych informacji o podatnościach [49], zatem równoważność punktacji może nie być osiągalna.



Rys. 16. Miary podatności do przeprowadzenia analizy podatności w standardzie CWSS. Źródło: [49]

- **CCSS (*Common Configuration Scoring System*)**

Mechanizm służący do badania stopnia zagrożenia podatności, wynikających z zastosowania niepoprawnej konfiguracji systemu [60]. Przykłady zidentyfikowanych wad w konfiguracji systemów gromadzone są w oficjalnym słowniku CCE (ang. *Common Configuration Enumeration*);

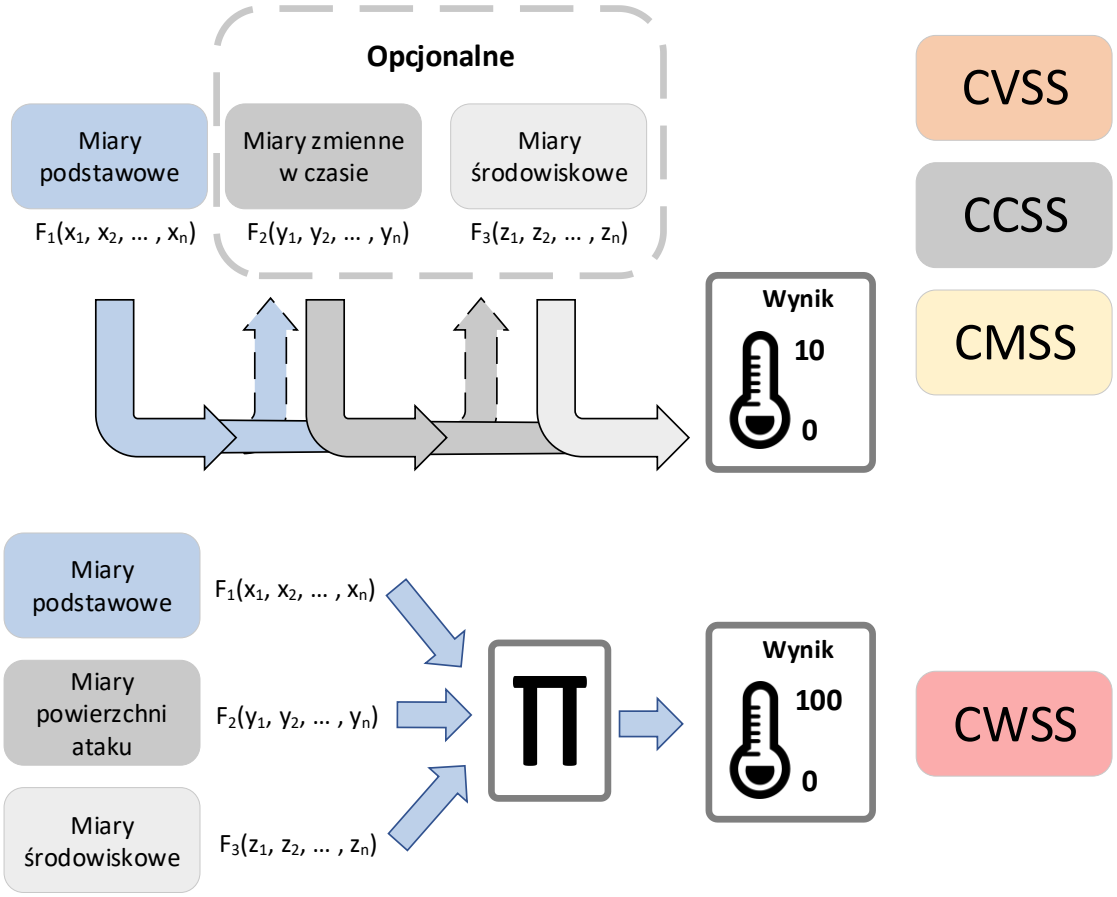


Rys. 17. Miary podatności zastosowane w standardach CCSS i CMSS. Źródło: [60] [61]

- **CMSS (*Common Misuse Scoring System*)**

Mechanizm służący do pomiaru stopnia zagrożenia podatności, pochodzących z niewłaściwego użytkowania oprogramowania, którego funkcjonalności mogą być wykorzystane w niepożądany sposób, m.in. do przeprowadzenia ataku na system [61]. Klasycznym przykładem może być odbiór złośliwego oprogramowania przesłanego za pomocą poczty internetowej lub podążanie za linkami przekierowującymi na zainfekowane witryny internetowe, itp. Wiele incydentów cybernetycznych związanych z niewłaściwym użytkowaniem funkcji, które daje oprogramowanie, jest następstwem wynikającym z zastosowania technik inżynierii społecznej przez atakującego.

W standardach CVSS, CCSS oraz CMSS wyróżniamy trzy grupy miar - miary podstawowe (ang. *Base Metrics*), zmienne w czasie (ang. *Temporal Metrics*) oraz miary środowiskowe (ang. *Environmental Metrics*). Miary podstawowe określają charakterystyki badanej podatności, które nie powinny ulegać zmianom w czasie, oraz nie są zależne od środowiska, w którym dana podatność występuje. Miary zmienne w czasie dotyczą właściwości podatności, które mogą się zmieniać wraz z upływem czasu, wskazujące np. poziom dojrzałości kodu podatności (ang. *Exploit Code Maturity*) czy poziom zaawansowania prac nad wykonaniem łatki dla podatności (ang. *Remediation Level*). Miary środowiskowe łączą w sobie te właściwości podatności z miar podstawowych oraz zmiennych w czasie, które zależne są również od systemu lub oprogramowania w którym podatność ta występuje. Warto zwrócić uwagę na to, iż zarówno miary zmienne w czasie jak i środowiskowe nie są obligatoryjne podczas procesu tzw. „scoringu", natomiast pozwalają na bardziej szczegółowy pomiar poziomu zagrożenia stwarzanego przez podatność. W standardzie CWSS, zastosowano podobne podejście do budowy wektora podatności jak w standardach CVSS, CCSS i CMSS, jednakże jest on bardziej rozbudowany i pozwala na zadawanie własnych wartości parametrów. Jednocześnie zauważalne są różnice w podziale miar na grupy, gdyż poza miarami podstawowymi i środowiskowymi CWSS należy uwzględnić charakterystyki podatności związane ściśle podłożem ataku, który może być skutkiem jej wykorzystania przez atakującego. Dodatkowo, w niektórych przypadkach CWSS umożliwia pominięcie wybranych składowych wektora podatności [49].

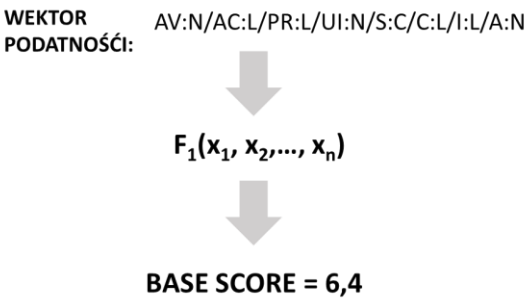


Rys. 18. Schemat procesu „scoringu" dla standardów CVSS, CCSS, CMSS oraz CWSS. Źródło: Opracowanie własne.

Zasada działania wspomnianych systemów „scoringowych" jest dla każdego standardu podobna. W celu wykonania badania, wartości składowych wektora podstawiane są do wzorów równań określonych przez standard dla każdej grupy miar, a zwracany wynik mieści się w przedziale [0; 10] dla CVSS, CCSS, CMSS lub [0; 100] w przypadku CWSS. Im mniejsza wartość zostanie zwrócona, tym w mniejszym stopniu groźna jest analizowana podatność. Schemat koncepcyjny badania stopnia zagrożenia podatności przedstawiono na Rys. 18. W podejściu inżynierskim wektor zapisuje się jako ciąg znaków. Wzory równań $F_n(x_1, x_2, \dots, x_n)$ przeznaczone do wyliczenia wartości stopnia zagrożenia stwarzanego przez podatność oraz wartości składowych wektora podatności dla wszystkich wspomnianych systemów „scoringowych" zostały ustalone metodą ekspercką. Schemat wyliczania wartości metryki podstawowej ze standardu CVSS 3.0 zaprezentowano na rysunku Rys. 19, na przykładzie podatności *SQL Injection for MySQL* (*CVE-2013-0375*).

MIARA	SKRÓT	Wartość (zm. lingwistyczna)	Wartość liczbowa
Attack Vector	AV	Network	0,85
Attack Complexity	AC	Low	0,77
Priviliges Required	PR	Low	0,68 *
User Interaction	UI	None	0,85
Scope	S	Changed	-
Confidentiality Impact	C	Low	0,22
Integrity Impact	I	Low	0,22
Availability Impact	A	None	0

* Gdy dla miary **Scope** ustalono wartość „Changed" miara **Priviliges Required** dla wartości „Low" przyjmuje wartość liczbową 0,68 (0,62 w p.p.)

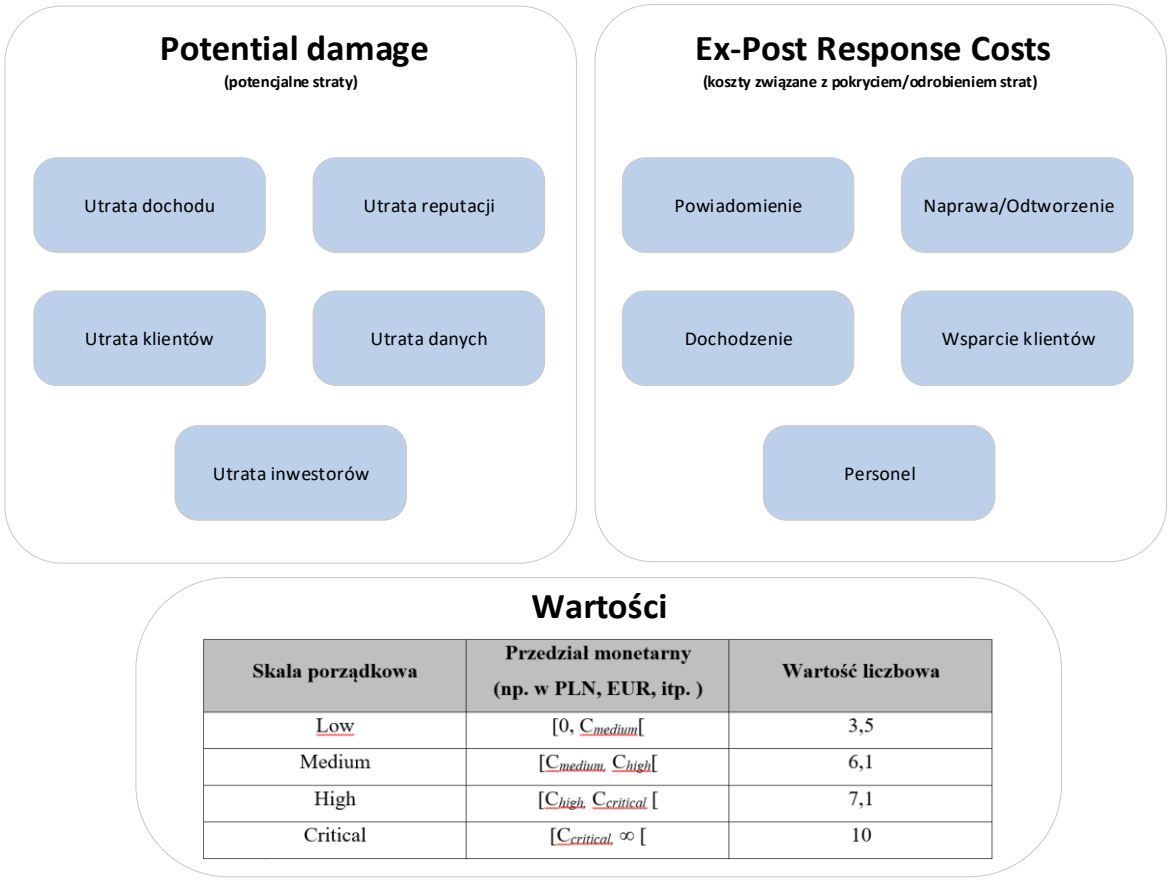


Rys. 19. Przykład wyliczenia metryki podstawowej w standardzie CVSS 3.0 dla podatności, pozwalającej na przeprowadzenie ataku SQL Injection w MySQL. Źródło: Opracowanie własne.

Pomiar za pomocą wspomnianych mechanizmów może być wykonywany w sposób automatyczny, przy wykorzystaniu np. dedykowanych kalkulatorów lub ręczny. Jednakże należy zaznaczyć, że istnieje wiele czynników, dzięki którym pomiar automatyczny może być nie zawsze dokładny, stąd zaleca się aby przeprowadzany był w kilku turach. Dla przykładu pierwsze próby obliczeń wykonane mogą być w narzędziu, po czym ekspert z dziedziny bezpieczeństwa uzupełnia parametry pomiaru i wykonuje go ponownie.

2.2.1.2 Przykład wykorzystania zmodyfikowanej wersji standardu CVSS

Ilościowe metody służące do pomiaru stopnia zagrożenia podatności, zastosowane w opisanych w poprzednim podrozdziale standardach, umożliwiają ustalanie priorytetów eliminowania podatności w badanym oprogramowaniu, przede wszystkim z punktu widzenia czysto-technicznego [62]. W literaturze światowej znaleźć można jednak przykłady ich modyfikacji, ponieważ powszechne wykorzystanie domyślnie zdefiniowanych w nich miar często nie obejmuje w pełni aspektu związanego z kontekstem wystąpienia podatności m. in. na poziomie organizacji. Mowa tutaj np. o scenariuszach, w których zauważalny jest brak uwzględnienia skutków ekonomicznych, jakie wywołać może wykorzystanie badanych podatności. W konsekwencji ocena stopnia zagrożenia podatności za pomocą podstawowych miar uwzględnionych w standardzie może nie być w konkretnych przypadkach miarodajna. Przykładem rozszerzenia standardu CVSS, rozszerzającego kontekst badania jest podejście, w którym zastosowano dwie dodatkowe grupy miar, odzwierciedlające skutki ekonomiczne [62], które przedstawiono na rysunku Rys. 20.

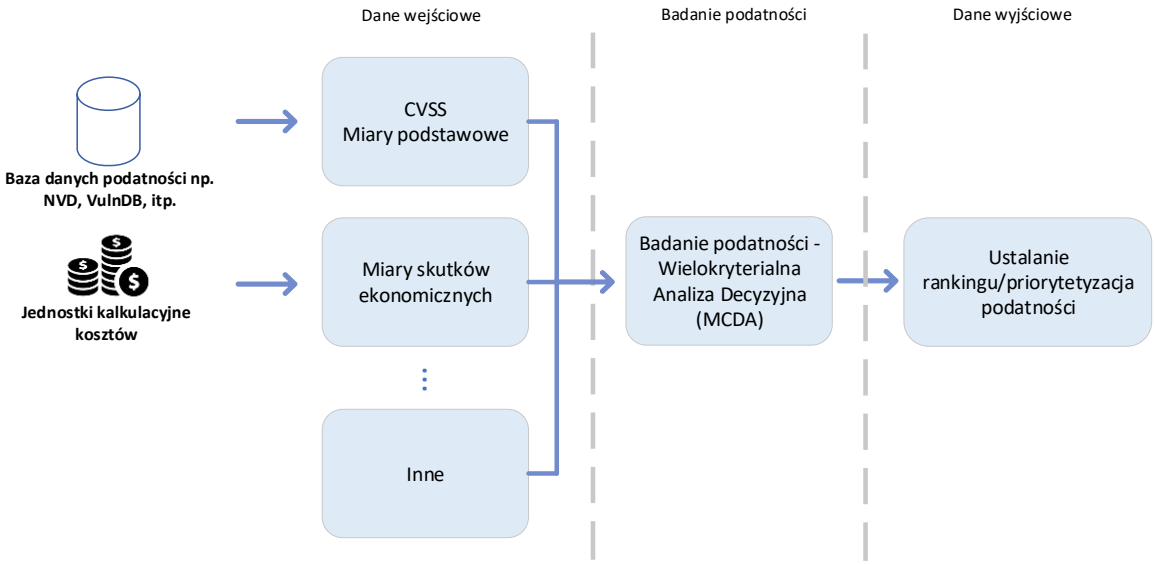


Wartości

Skala porządkowa	Przedział monetarny (np. w PLN, EUR, itp.)	Wartość liczbowa
Low	$[0, C_{medium}[$	3,5
Medium	$[C_{medium}, C_{high}[$	6,1
High	$[C_{high}, C_{critical}[$	7,1
Critical	$[C_{critical}, \infty[$	10

Rys. 20. Ekonomiczne miary podatności zastosowane w zmodyfikowanej wersji standardu CVSS. Źródło: [62]

Pierwsza grupa obejmuje miary, charakteryzujące potencjalne straty dla organizacji, będące wynikiem wykorzystania podatności w celu skompromitowania ich systemów. Zaliczamy do nich straty dochodu, klientów, inwestorów, danych oraz reputacji. Druga grupa miar w zaproponowanym podejściu służy do wskazania pięciu rodzajów kosztów związanych z naprawą strat po potencjalnym wykorzystaniu podatności przez atakującego [62] [63]. W nawiązaniu do standardu CVSS, do ustalenia wartości miar przyjęto czterostopniową skalę – *Low, Medium, High, Critical*, przypisując wymienionym zmiennym lingwistycznym konkretne wartości od 0 do 10. Z uwagi na fakt występowania różnic w charakterystykach organizacji, w tym m. in. wielkości ich budżetów założono, iż należy znaleźć odzwierciedlenie wartości miar, stosując przy tym przedziały zadawane w jednostkach monetarnych. Wynikiem tego jest zdolność definiowania własnych wartości przedziałów przez każdą z organizacji, wykorzystującą miary ekonomiczne w celu wykonania badania podatności.



Rys. 21. Schemat poglądowy realizacji procesu analizy i ustalania rankingu podatności dla przykładu zmodyfikowanej wersji standardu CVSS. Źródło: [62]

Proces badania podatności, wykorzystujący zdefiniowane miary ekonomiczne, oparty jest na wielokryterialnej analizie decyzyjnej (ang. *Multiple Criteria Decision Analysis*), realizowanej z pomocą metody matematycznej MAHP (*Multiplicative Analytic Hierarchy Process*). W metodzie tej, dla zbioru m podatności v_i (dla $i = 1,2,\dots,m$), zidentyfikowanych dla badanego oprogramowania, przy założeniu n kryteriów decyzyjnych c_j (dla $j = 1,2,\dots,n$), gdzie dla każdego kryterium decyzyjnego określona jest waga w_j (dla $j = 1,2,\dots,n,\ \sum_{j=1}^{n} w_j = 1$) możliwe jest ustalenie priorytetów podatności, poprzez utworzenie rankingu. Kryteriami są w tym przypadku podstawowe miary ze standardu CVSS, miary ekonomiczne, będące jego rozszerzeniem, a także opcjonalnie, dodatkowe miary zdefiniowanych przez testującego. Wzór do obliczania wyniku dla pojedynczego egzemplarza podatności oraz macierz oceny kryteriów przedstawiono na schemacie Rys. 22

$$\begin{array}{c} \\ \\ \textit{Podatności} \end{array} \begin{array}{c} \textit{Kryteria} \\ \begin{array}{cccc} C_1 & C_2 & \cdots & C_n \end{array} \\ \begin{array}{cccc} (w_1 & w_2 & \cdots & w_n) \end{array} \\ \begin{array}{c} v_1 \\ v_2 \\ \cdots \\ v_m \end{array} \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \end{array} \Rightarrow P_{v_i} = \prod_{j=1}^{n} (a_{ij})^{w_j}$$

Rys. 22. Macierz oceny kryteriów dla metody MAHP w procesie badania podatności uwzględniającego miary ekonomiczne. Źródło: [62]

gdzie a_{ij} oznacza wartość miary podatności oznaczonej pod kryterium j dla podatności i. Przykładem mogą być wartości miar obliczone w systemie scoringowym CVSS, t.j. miara podstawowa, miara środowiskowa, miara zmienna w czasie, wspomniane w podrozdziale 2.2.1.1 oraz ww. miary skutków ekonomicznych.

2.2.2 Modele wykrywania podatności

Obecnie ilościowe metody badania niezawodności oprogramowania są wykorzystywane na porządku dziennym. Bezpieczeństwo oprogramowania również potrzebuje podobnego podejścia, aby możliwe było skuteczniejsze przeciwdziałanie realizacji cyberzagrożeń. W przypadku badania zbiorów podatności oprogramowania, analogicznie do podejścia stosowanego w badaniu niezawodności oprogramowania opracowano matematyczne modele – *Vulnerability Discovery Models* (VDM), przeznaczone do wsparcia prognozowania procesu wykrywania podatności. Zostały one utworzone korzystając ze zdobyczy modelowania matematycznego opierając się na danych aktualnych oraz danych historycznych pozyskanych w trakcie badań oprogramowania. Dzięki takiemu podejściu możliwe jest prognozowanie liczby potencjalnych podatności, które mogą zostać wykryte w przyszłości. Precyzyjnie zdefiniowany model stanowi użyteczną wiedzę do szacowania bezpieczeństwa oprogramowania, ale może być również dobrym narzędziem dla użytkowników jak i twórców oprogramowania, służącym do prognozowania trendów bezpieczeństwa oprogramowania, jak i planowania szeroko rozumianego procesu zarządzania jego bezpieczeństwem [64] [65].

W literaturze znaleźć można niewiele prac dotyczących modeli wykrywania podatności. Dla dostępnych modeli VDM wyróżnić można dwie zasadnicze kategorie – *Time-Based Models* oraz *Effort-Based Models*. Pierwsza grupa modeli służy do predykcji łącznej liczby podatności w funkcji czasu (ang. *time as governing factor*) [56], druga grupa umożliwia predykcję łącznej liczby podatności w zależności np. od udziału oprogramowania w rynku, czy liczby użytkowników Większość istniejących modeli to modele w funkcji czasu, bazujące na bardzo różnych założeniach, przez co można zauważyć różnice w skuteczności i trafności oferowanej przez nie prognozy. Wyniki analizy przy wykorzystaniu modeli VDM mogą być wykorzystane do wyznaczania metryk bezpieczeństwa oprogramowania np. (ang. *Vulnerability Density*) czy wskaźnika wykrywalności podatności (ang. *Vulnerability Discovery Rate*) [66]. W kolejnych

podrozdziałach zaprezentowano krótką charakterystykę wybranych modeli wykrywania podatności.

2.2.2.1 Liniowe modele VDM

W literaturze znaleźć można dwa rodzaje modeli liniowych VDM:

- Liniowy prosty (ang. *Simple Linear - LN*);

- Model liniowy autorstwa E. Rescorli (*Rescorla Quadratic - RQ*).

Model liniowy prosty wskazuje liniowy trend wykrywania podatności i określony jest następującym wzorem funkcji liniowej [56]:

$$N(t) = a \cdot t + k. \tag{2.1}$$

gdzie $N(t)$ to liczba wszystkich wykrytych podatności do chwili t, k jest stałą, a współczynnik kierunkowy prostej a to wskaźnik wykrywalności podatności.

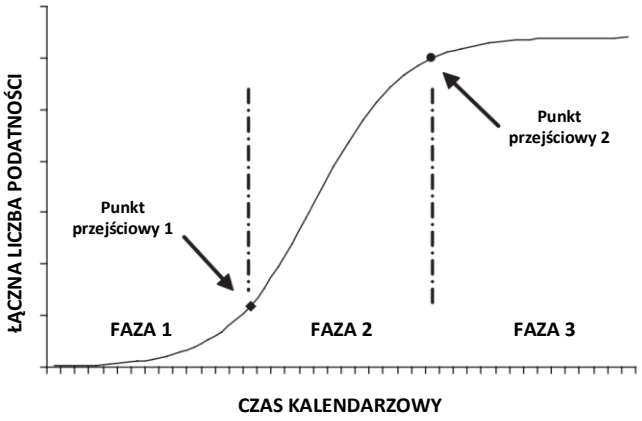
Model liniowy E. Rescorli zakłada, że liczba wszystkich wykrytych podatności $N(t)$ w badanym oprogramowaniu oparta jest na kwadratowej zależności, przedstawionej następująco [67]:

$$N(t) = \frac{B \cdot t^2}{2} + k \cdot t + c \tag{2.2}$$

gdzie B i k są współczynnikami regresji oraz B determinuje współczynnik nachylenia krzywej ilustrującej liczbę wykrytych podatności w funkcji czasu.

2.2.2.2 Model AML

AML to logistyczny model czasowy autorstwa Alhazmi i Malaiya [56], dla którego wyróżniono trzy zasadnicze fazy wykrywania podatności, jak zilustrowano na Rys. 23.



Rys. 23. Trzy fazy wykrywania podatności oprogramowania dla modelu AML. Źródło: [56]

W pierwszej fazie, zwanej fazą uczenia (ang. *learning phase*), następuje stopniowy wzrost liczby użytkowników, w tym m.in. testerów bezpieczeństwa oraz potencjalnych atakujących. System znajduje się na etapie wdrożenia, stąd zaobserwować można stosunkowo niewielki wzrost liczby wykrytych w nim podatności. Po pewnym czasie (osiągając punkt przejściowy nr 1 na Rys. 23) liczba użytkowników, zarówno jak i testujących oprogramowanie wzrasta, przez co odnotowuje się liniowy wzrost wykrytych podatności - system znajduje się wtedy w tzw. fazie liniowej (ang. *linear* phase). Następnie gdy wdrażany jest system będący następnikiem dla aktualnie badanego, ten znajduje się w fazie nasycenia (ang. *saturation phase*) (punkt przejściowy nr 2 na Rys. 23), od której wskaźnik wykrywalności podatności maleje.

Model AML bazuje na założeniu że tempo zmian (ang. *rate of change*) liczby wszystkich podatności $N(t)$, wykrytych do chwili t, zależy od dwóch współczynników: pierwszy z nich maleje wraz ze spadkiem liczby pozostałych- nieodkrytych podatności w systemie, podczas gdy drugi współczynnik rośnie wraz z czasem od rozpoczęcia działania systemu. Przy zastosowaniu powyższych założeń – wskaźnik wykrywalności podatności określono następującym równaniem różniczkowym [56]:

$$\frac{dN(t)}{dt} = bN(t)(a - N(t)) \tag{2.3}$$

gdzie $N(t)$ to skumulowana liczba podatności do chwili t, a i b są stałymi wyznaczonymi empirycznie, przy czym b interpretuje się jako intensywność wykrywania podatności. Po rozwiązaniu równania (2.3) otrzymujemy wzór na liczbę wszystkich podatności wykrytych do chwili t [56]:

$$N(t) = \frac{a}{ace^{-abt} + 1} \tag{2.4}$$

gdzie: c jest stałą wprowadzoną podczas rozwiązywania równania (2.3). Z równania (2.4) wynika, że podczas gdy t dąży do nieskończoności, $N(t)$ dąży do a. Stąd łączna liczba wykrytych podatności jest reprezentowana przez parametr a.

2.2.2.3 Model na bazie stochastycznego równania różniczkowego Itô

Kolejny Model VDM – model stochastyczny, może być uznany za udoskonalenie modelu AML. Motywacją dla autorów do jego utworzenia był fakt, mówiący iż wiele z istniejących modeli wykrywania podatności opartych jest na założeniach, często bardzo teoretycznych, nieuwzględniających prawdziwych realiów branżowych, stąd trudno je

w rzeczywistości zastosować np. w przemyśle. Do jego budowy wprowadzono kilka dodatkowych parametrów, które nie zostały uwzględnione w modelu AML, istotnych z punktu widzenia możliwości wystąpienia potencjalnych podatności na zagrożenia w oprogramowaniu. Sformułowano je następująco [68]:

1. Proces wykrywania podatności jest procesem stochastycznym z ciągłą przestrzenią stanów.
2. Wraz z postępem w badaniach, liczba pozostałych podatności w wykazuje tendencję spadkową.
3. Może wystąpić awaria systemu podczas jego działania, jako że posiada podatności.
4. Proces eliminacji podatności jest idealny i podczas tejże eliminacji nie powstają, żadne nowe podatności.

Dodatkowo, przyjęto, iż zmienna losowa reprezentująca liczbę podatności wykrytych w systemie, aż do czasu t określonego podczas badania –$N(t)$ może przyjmować ciągłe liczby rzeczywiste. W przypadku przyjęcia odpowiednio dużego rozmiaru systemu, liczba podatności wykrytych podczas badania jest również duża, to zmiana liczby podatności wykrytych i usuniętych podczas każdorazowego „debugowania" jest wystarczająco mała w porównaniu do liczby podatności ze zbioru wejściowego. Po przyjęciu powyższego założenia autorzy zamodelowali proces wykrywania podatności w systemie jako stochastyczny proces z ciągłą przestrzenią stanów. Podążając za postępem badania liczba pozostałych podatności w systemie będzie się zmniejszać, jako że tzw. „uśpione" podatności będą wykrywane i eliminowane z systemu. W rezultacie, równanie różniczkowe przyjmuje następującą postać [68]:

$$\frac{dN(t)}{dt} = b(t)[a - N(t)] \tag{2.5}$$

gdzie $b(t)$ jest to stosunek liczby wykrytych podatności do liczby podatności pozostałych w systemie oraz funkcja opisująca proces wykrywania podatności przyjmuje wartości nieujemne. Zauważono, iż badanie podatności w systemie jest zależne od wielu czynników, a nie każdy z nich jest deterministyczny, jak na przykład wydatki, efektywność testowania, umiejętności testerów, metody i strategie testowania. Rozważając te niedeterministyczne czynniki, mamy świadomość, że wpływają one na wynik badania w tym wypadku liczbę wykrytych i wyeliminowanych podatności - $b(t)$.

Następnie przekształcono równanie (2.5) na takie, które odzwierciedla właściwości stochastycznego procesu testowania, a zarazem odpowiadające bardziej realistycznemu scenariuszowi badania podatności. W tym celu, zakładając nieregularne odchylenia w $b(t)$, rozszerzono je poprzez randomizację do następującego stochastycznego równania różniczkowego [68]:

$$\frac{dN(t)}{dt} = \{b(t)+\sigma\gamma(t)\}\{a - N(t)\} \tag{2.6}$$

gdzie $\sigma\gamma(t)$ jest standaryzowanym białym szumem gaussowskim, $\gamma(t) = \frac{dW(t)}{dt}$, a $W(t)$ jest procesem Wienera [69].

Dalsze przekształcenia, uwzględniające zastosowanie formuły Itô, pozwoliły autorom określić wzór na oczekiwaną liczbę podatności wykrytych w przedziale $(0;\ t]$ wyrażoną jako [68]:

$$m(t) = E[N(t)] = a \cdot \left(1 - \frac{\left(\frac{a-k}{k}\right) \cdot e^{-(a \cdot b \cdot t - \frac{1}{2}\cdot\sigma^2 \cdot t)}}{\left(1 + \left(\frac{a-k}{k}\right) \cdot e^{-a \cdot b \cdot t}\right)}\right) \tag{2.7}$$

2.2.2.4 Model oparty o rozkład Weibull'a

Model VDM zbudowany w oparciu rozkład Weibulla zakłada, że wskaźnik wykrywalności podatności ω zmienia się w zależności od funkcji gęstości prawdopodobieństwa dla tego rozkładu, co wyraża się następującym wzorem [70]:

$$\omega(t) = \gamma \cdot \left\{\frac{\alpha}{\beta} \cdot \left(\frac{t}{\beta}\right)^{\alpha-1} \cdot e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\right\} \tag{2.8}$$

gdzie α jest parametrem kształtu rozkładu, który determinuje wskaźnik wykrywalności podatności. W przypadku gdy wartość parametru α wynosi w przybliżeniu 3, wówczas kształt jest symetryczny [70]. Skośność krzywej przyjmuje ujemne wartości gdy $\alpha > 3$ natomiast dodatnie w p. p.. Skalujący parametr β pozwala zwiększyć zakres czasu t przyjętego dla modelu, natomiast γ wskazuje liczbę podatności które w danym czasie jeszcze zostaną wykryte [71]. Z przekształceń równania (2.8) wyznaczyć można równanie służące do określenia liczności zbioru wszystkich podatności $N(t)$- dla badanego oprogramowania, zapisane następująco [70]:

$$N(t) = \gamma \cdot \{1 - \cdot\, e^{-\left(\frac{t}{\beta}\right)^{\alpha}}\} \tag{2.9}$$

2.2.2.5 Model wykładniczy autorstwa E. Rescorla

Wykładniczy model wykrywania podatności autorstwa E. Rescorla [67] to model czasowy, w budowie którego autor opiera się na założeniach modelu wzrostu niezawodności oprogramowania Goel-Okumoto [72], analizując dane historyczne badanego oprogramowania. Zdefiniowany został następująco [67]:

$$n(t) = N \cdot \lambda \cdot e^{-\lambda t} \tag{2.10}$$

gdzie N jest stałe i definiuje maksymalną liczbę podatności w badanym oprogramowaniu, a λ jest stałym współczynnikiem służącym do określenia stopy wykrywania podatności oprogramowania [69]. Po scałkowaniu równania (2.10) otrzymujemy skumulowaną liczbę podatności w oprogramowaniu - $N(t)$, wykrytych w funkcji czasu, wyrażoną następującym wzorem [67]:

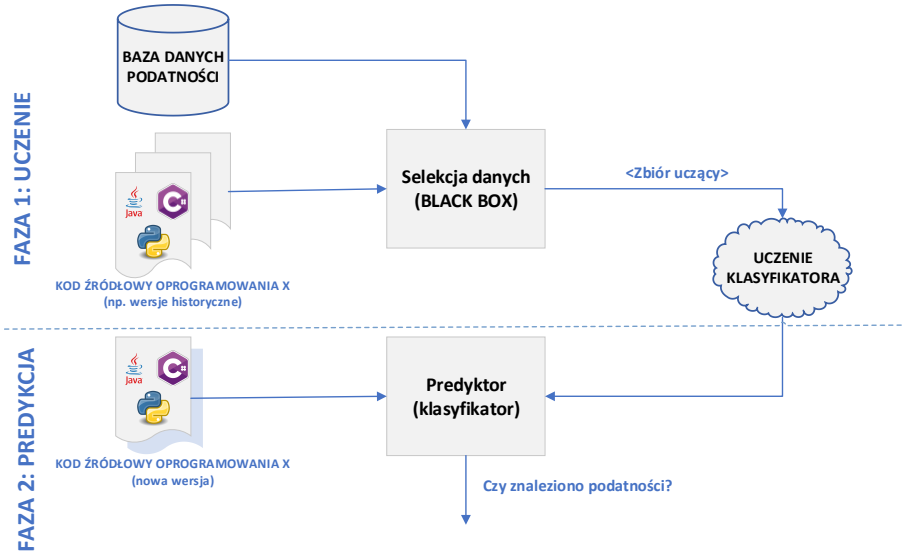
$$N(t) = N \cdot (1 - e^{-\lambda \cdot t}) \tag{2.11}$$

2.2.3 Modele predykcji podatności

Modelowanie predykcji podatności (ang. *Vulnerability Prediction Modeling*) to kolejny obszar obok modeli VDM, mający na celu zwiększenie poziomu bezpieczeństwa badanego oprogramowania przy uwzględnieniu występowania podatności. Powstałe w obrębie tej dziedziny modele predykcji podatności oprogramowania VPM (ang. *Vulnerability Prediction Models*) oparte są na technikach uczenia maszynowego [73], a wykorzystywane do ich konstrukcji metody klasyfikacji danych służą do prognozy ewentualnego wystąpienia podatności w przyszłych wersjach poszczególnych komponentów badanego oprogramowania.

Zagadnienie klasyfikacji danych dotyczy pewnych obiektów opisywanych przez swoje cechy i należących do pewnych klas. Polega na skonstruowaniu klasyfikatora, czyli funkcji, która na podstawie opisu obiektu oblicza jego klasę. Proces klasyfikacji można podzielić na dwa etapy: uczenia się, w którym następuje wygenerowanie wiedzy niezbędnej do przeprowadzenia procesu, oraz etap właściwego wyznaczania wyniku, w trakcie którego używana jest wiedza wygenerowana w fazie uczenia [74]. Do rozwiązania zadania klasyfikacji wykorzystywany jest szereg technik i algorytmów uczących, pośród których zaobserwować można istotne różnice np. w zasadzie działania, złożoności, rodzaju danych wejściowych, co może mieć wpływ na jakość modelu klasyfikacji. W pracach nad modelami VPM autorzy wykorzystali w swoich rozwiązaniach m. in. naiwny klasyfikator Bayesa, lasy

losowe (ang. *Random Forest*), sieci neuronowe, regresję logistyczną czy maszynę wektorów nośnych - *SVM* (z ang. *Support Vector Machine*) [75].

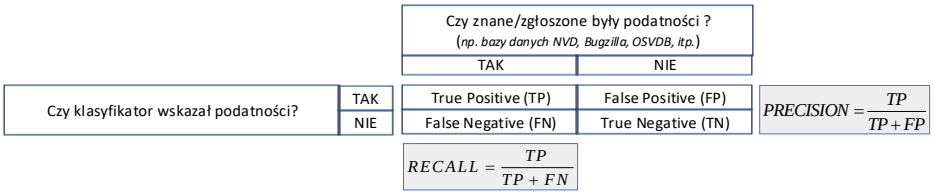


Rys. 24. Podstawowy schemat uczenia maszynowego dla modeli VPM. Źródło: [76]

Dla zilustrowania procesu klasyfikacji modeli VPM w ujęciu ogólnym, niech C oznacza analizowany komponent oprogramowania. Według założeń przyjętych przez autorów modeli VPM komponentami są najczęściej pojedyncze pliki klas oprogramowania wchodzące w skład projektu [75] [77] [78] [79] [80]. Ponadto, komponent opisywany przez pewną liczbę ustalonych cech $c_1, c_2, \ldots, c_n$, gdzie n jest liczbą rozpatrywanych cech. W tym przypadku cechami komponentów mogą być m.in. wartości wybranych metryk oprogramowania, charakterystyki poszczególnych fragmentów kodu źródłowego, informacje o wersjach historycznych i wiele innych. Dla uproszczenia zapisu przyjąć można, iż komponent oprogramowania zapisać można jako wektor cech $C = [c_1, c_2, \ldots, c_n]$. W procesie klasyfikacji wyznaczana jest przynależność każdego z badanych obiektów do jednej z pewnej liczby ustalonych klas $K_1, K_2, \ldots, K_m$, gdzie m jest liczbą rozpatrywanych klas. Ze znanych z literatury modeli VPM wynika, iż są one klasyfikatorami binarnymi- $m = 2$ [75] [77] [78] [79] [80]. W rozpatrywanych przypadkach, ta tzw. klasyfikacja dwuklasowa pozwala w rezultacie odpowiedzieć na pytanie do której z dwóch dostępnych klas należy komponent oprogramowania: K_1- zawierający lub K_2- niezawierający podatności.

Do oceny zdolności predykcyjnych zaproponowanego modelu, w przypadku klasyfikacji binarnej można posłużyć się wartościami zapisanymi w tablicy pomyłek (macierzy błędów), w której wiersze przedstawiają klasy predykowane, kolumny zaś klasy

rzeczywiste (Rys. 25). Na przecięciu kolumn i wierszy znajdują się cztery wartości liczbowe, spośród których wartości oznaczone jako True Positive (TP) i True Negative (TN) wskazują liczbę wyników pozytywnych i negatywnych zwróconych przez klasyfikator, natomiast False Positive (FP) i False Negative (FN) określają ile pozytywnych i negatywnych przypadków zostało sklasyfikowanych w sposób niewłaściwy.



Rys. 25. Schemat poglądowy obliczania wartości miar jakości: precyzja i wrażliwość dla modeli VPM. Źródło: [76]

W oparciu o wartości poszczególnych komórek macierzy można w konsekwencji wyznaczyć wybrane miary zdolności predykcyjnych takie jak np. precyzja (ang. *Precision,* lub *positive predictive value - PPV*) i wrażliwość (ang. *Sensitivity* lub *recall,* lub *true positive rate - TPR*). Precyzja w przypadku modeli VPM określa jaka część wykrytych podatności jest prawdziwa. Wyrażana jest jako stosunek liczby poprawnie wykrytych podatności do liczby wszystkich pozytywnych wyników otrzymanych z predykcji modelu:

$$precyzja = \frac{t_p}{t_p + f_p} \tag{2.12}$$

Wrażliwość natomiast wskazuje jaka część podatności została wykryta przez klasyfikator i wyraża się ją jako iloraz liczby trafnie zdiagnozowanych przypadków wystąpienia podatności oraz liczby wszystkich podatności:

$$wrażliwość = \frac{t_p}{t_p + f_n} \tag{2.13}$$

W dokonanym przeglądzie literatury zaobserwowano, iż najczęściej spotkać można dwa szersze obszary tematyczne [73], wokół których wytworzono modele VPM, powiązane z:

- analizą metryk oprogramowania;

- zastosowaniem metod eksploracji tekstu (ang. *text-mining*);

Przegląd oraz wybrane informacje dotyczące przeanalizowanych modeli VPM wraz z ich krótką charakterystyką zostały przedstawione w kolejnych podrozdziałach.

2.2.3.1 Modele VPM budowane w oparciu o metryki oprogramowania

Pierwsza grupa modeli predykcji podatności VPM obejmuje prace wykorzystujące zmierzone wartości metryk oprogramowania. Motywem jakim posłużyli się autorzy takiego rodzaju rozwiązań mógł być sukces jaki analogiczne podejścia odniosły w prognozowaniu defektów oprogramowania [75], a dodatkowo przy ich budowie początkowo zakładano, iż pozwolą w sposób doświadczalny wypracować opinię ekspertów do spraw bezpieczeństwa, że złożoność oprogramowania ma znaczący wpływ na jego bezpieczeństwo.

2.2.3.1.1 Model „CCD"

Model CCD to model predykcji podatności VPM bazujący na trzech grupach metryk oprogramowania, które z powodzeniem wykorzystano do prognozowania defektów oprogramowania. Są to metryki służące do określenia jego złożoności (ang. *complexity*), związane ze zmianami kodu (ang. *code churn*), oraz określające aktywność programisty (ang. *developer activity metrics*) [77] [81].

Autorzy w badaniu wykorzystali technikę klasyfikacji opartą o regresję logistyczną, która opiera się na specyficznym sposobie wyrażania prawdopodobieństwa, zwanym szansą (ang. *odds*), czyli stosunkiem prawdopodobieństwa sukcesu do prawdopodobieństwa porażki. Wykorzystano funkcję przekształcającą prawdopodobieństwo na logarytm szansy zwaną logitem, którą można definiować następująco:

$$p = \frac{e^{log\,it(p)}}{1 + e^{log\,it(p)}} = \frac{1}{1 + e^{-log\,it(p)}} \tag{2.14}$$

gdzie p to prawdopodobieństwo wystąpienia zdarzenia, z to liniowa kombinacja zmiennych niezależnych x_i, a β_i to współczynnik regresji logistycznej dla i -tej zmiennej niezależnej. Badane oprogramowanie potraktowano jako zbiór plików i ustalono, iż każdy plik może zostać uznany jako podatny na zagrożenia z cyberprzestrzeni, jeśli wspomniane prawdopodobieństwo jest większe niż 0,5 [77]. Ponadto, zaobserwowano, iż tworzenie modelu VPM dla każdej z metryk oprogramowania indywidualnie skutkuje uzyskaniem niskich możliwości predykcyjnych [82]. Zdecydowano się zatem na utworzenie czterech rodzajów modeli – odpowiednio dla każdej z poszczególnych grup wspomnianych metryk jeden będący ich kombinacją.

Eksperyment przeprowadzono na oprogramowaniu przeglądarki Mozilla Firefox oraz oprogramowaniu jądra systemu operacyjnego Red Hat Enterprise Linux 4 Kernel. Do

realizacji przyjętych założeń wykorzystano opensource'owe środowisko ułatwiające pracę z algorytmami *uczenia maszynowego* WEKA 3.7 (*Waikato Environment for Knowledge Analysis*) [5]. W przypadku Mozilli Firefox, pozyskano dane o podatnościach z bazy danych Mozilla Foundation Security Advisories (*MFSA*)[6] dla 34 wersji wypuszczonych do użytkowania (wersje Mozilla Firefox 1.0 – 2.0.0.16). Podzielono je na jedenaście podzbiorów $\{R_1,\ R_2,\ \dots\ , R_{11}\}$, z których każdy, poza jednym wyjątkiem, zawierał dane o podatnościach z trzech kolejnych wersji przeglądarki. W celu sprawdzenia możliwości predykcji podatności oferowanej przez model przeprowadzono następnie walidację opartą o dane z otrzymanych zbiorów. Podejście wykorzystane do walidacji modelu polegało na utworzeniu zbioru uczącego przy użyciu danych z trzech kolejnych podzbiorów np. $\{R_1, R_2, R_3\}$, a kolejny podzbiór, w tym przypadku $\{R_4\}$ stanowił zbiór testowy (tzw. ang. *next-release validation*). Z uwagi na fakt, iż wykorzystanie zbyt dużej liczby metryk oprogramowania jednocześnie często obniża skuteczność modelu [83], zdecydowano się na wykonanie selekcji cech przy pomocy metody *Information Gain*, a następnie na losowe usunięcie przykładów danych z klasy większościowej (wykonanie tzw. *undersamplingu*).

Dla oprogramowania RHEL4 natomiast, składającego się z 13568 plików w języku C, pozyskano dane o podatnościach z bazy danych *Red Hat Bugzilla*[7] oraz programu *RPM Package* Manager[8] dla wersji jądra systemu 2.6.0 – 2.6.9. Następnie przeprowadzono 10-krotną walidację krzyżową dla modelu wykorzystującego w założeniach regresję logistyczną. Do tego celu podzielono uzyskany zdane w sposób losowy na dziesięć podzbiorów, przy czym dla każdej iteracji dziewięć zbiorów wykorzystywano jako uczące, a pozostały stanowił zbiór testowy. Analogicznie jak dla przypadku Mozilli Firefox zdecydowano na selekcję danych metodą *Information Gain* i przefiltrowanie przy pomocy *undersampling'u* i ostatecznie, wyniki dla przeprowadzonej 10-krotnej walidacji krzyżowej dla RHEL4 zostały uśrednione.

Wykonano osiemdziesiąt prognoz bazując na danych o podatnościach przeglądarki Mozilla Firefox oraz sto dla oprogramowania jądra systemu Red Hat Enterprise Linux 4. W obu przypadkach uzyskano bardzo niską precyzję, dla przeglądarki wyniosła około 3%, a dla jądra RHEL4 – 5%. Spowodowane było to uzyskaniem względnie dużej liczby błędów pierwszego rodzaju (ang. *false positives*). W przypadku wskaźnika *Recall*, dla przeglądarki

[5] https://www.cs.waikato.ac.nz/ml/weka/
[6] https://www.mozilla.org/en-US/security/advisories/
[7] https://bugzilla.redhat.com/
[8] http://rpm.org/

był to wynik w przedziale 79-86%, a dla jądra systemu RHEL4 – w przedziale 80-90%. We wnioskach z przeprowadzonych badań autorzy wskazują również, iż utworzony model VPM został zweryfikowany na bazie tylko dwóch projektów, stąd jego miary podatności oprogramowania nie mogą być generalizowane i stanowić punktu odniesienia dla innego rodzaju oprogramowania, o różnej wielkości, a także projektów utworzonych w różnych technologiach [77].

2.2.3.1.2 Model autorstwa V.H. Nguyen i L. M. S. Tran

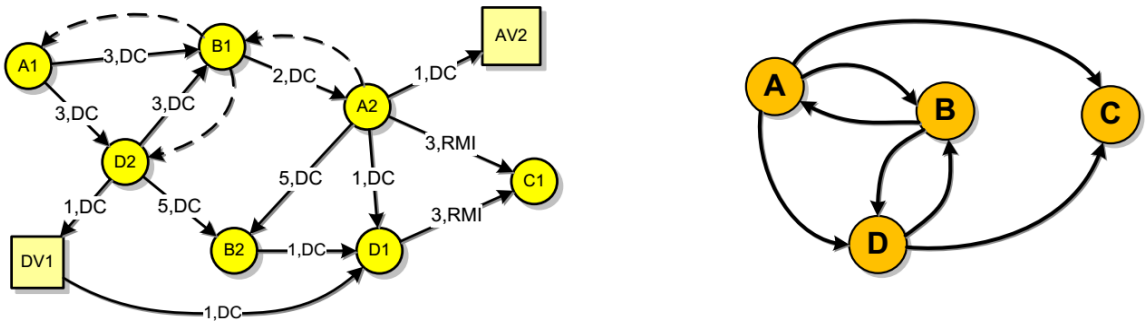
Model VPM opisany w pracy [76] zbudowany został przy wykorzystaniu autorskiej metryki bazującej na grafach zależności, opisujących relację pomiędzy komponentami oprogramowania, wyodrębnionymi przez statyczny analizator kodu. Formalnie przyjęto, że komponent oprogramowania to uporządkowana dwójka $\langle D_C, M_C \rangle$, gdzie D_C to zbiór zmiennych i danych składowych, natomiast M_C to zbiór metod i funkcji. Oprogramowanie, określone zostało jako czwórka $\langle S_C,\ D_S,\ M_S,\ R \rangle$, gdzie S_C to zbiór jego komponentów, $D_S = \bigcup_{C \in S_C}\ D_C$ to zbiór zmiennych i danych, $M_S = \bigcup_{C \in S_C}\ M_C$ to zbiór wszystkich funkcji. $R \subseteq (D_S \cup M_S) \times (D_S \cup M_S)$ jest to zbiór relacji pomiędzy składowymi oprogramowania, które podzielić można na cztery grupy, powiązane z: przekazywaniem parametrów, zwracaniem wartości funkcji, odczytem oraz zapisem danych. Oprogramowanie może być reprezentowane na dwa sposoby, przy wykorzystaniu dwóch rodzajów grafów zależności. Jednym z nich jest graf zależności składowych oprogramowania (ang. *Member Dependency Graph – MDG*), drugi to graf zależności komponentów oprogramowania (ang. *Component Dependency Graph – CDG*).

Graf MDG dla oprogramowania $S\ \langle S_C,\ D_S,\ M_S,\ R \rangle$ sformułowano jako graf skierowany $G_{MD}\ \langle V_M^d,\ V_M^m,\ E_M^c,\ E_M^r,\ E_M^d, \rangle$, gdzie $V_M^d \equiv D_S$ to zbiór wierzchołków będących składowymi związanymi z przepływem danych (ang. *data nodes*), $V_M^m \equiv M_S$ to wierzchołki reprezentujące zaimplementowane funkcje (ang. *function nodes*), $E_M^c \subseteq V_M^m \times V_M^m$ to krawędzie związane z wywoływaniem funkcji (tzw. *call-edges*), $E_M^r \subseteq V_M^m \times V_M^m$ to krawędzie odzwierciedlające zwracaniem wartości funkcji (tzw. *return-edges*) oraz $E_M^d \subseteq V_M^d \times V_M^m$ to krawędzie związane z dwukierunkową wymianą danych (tzw. *data-edges*).

Dla danego oprogramowania $S\ \langle S_C,\ D_S,\ M_S,\ R \rangle$ oraz zdefiniowanego dla niego grafu $G_{MD}\ \langle V_M^d,\ V_M^m,\ E_M^c,\ E_M^r,\ E_M^d, \rangle$, graf CDG sformułowano jako $G_{CD}\ \langle V_C,\ E_C \rangle$, dla którego $V_C \equiv S_C$ to zbiór wierzchołków reprezentujących komponenty oprogramowania,

a $E_C \subseteq S_C \times S_C$ jest to zbiór krawędzi, określających relacje między komponentami przy czym $E_C = \{\langle C_1, C_2\rangle |\ C_i \in S_C,\ \exists\langle m_1, m_2\rangle \in E_M \cdot m_i \in C_i,\ i = 1{,}2\}$ [76].

Przykładowe zobrazowanie grafów MDG oraz CDG dla komponentów oprogramowania realizujących metodę *Direct Call* w RMI[9] przedstawiono na Rys. 26.



Rys. 26. Przykład grafu zależności składowych oprogramowania reprezentujący realizację przypadku użycia „Direct Call" w RMI i wygenerowany na jego podstawie graf zależności komponentów oprogramowania. Źródło: [76]

Na potrzeby prognozowania, wierzchołkom oraz krawędziom przypisano wartości, uzyskane za pomocą statycznego analizatora kodu. Zastosowany w badaniu klasyfikator uczony jest za pomocą dwóch krotek $\langle r_{in}, r_{out}, oir, eir\rangle$ oraz $\langle r_{in}, r_{out}, V-density, I-density, oir, eir\rangle$, gdzie r_{in} i r_{out} oznaczają kolejno liczby krawędzi wchodzących i wychodzących z wierzchołków komponentów, oir to stosunek średnich wartości przepływu danych wchodzących i wychodzących z komponentu (ang. *ratio between average outgoing and incoming dataflow*), eir to stosunek uśrednionego przepływu danych w obrębie komponentu i poza nim (ang. *ratio between average internal and external dataflow*). $V-density$ określa złożoność cyklomatyczną McCabe'a komponentu oprogramowania, natomiast $I-density$ to złożoność jego interfejsu.

Eksperymenty weryfikujące skuteczność przyjętych założeń przeprowadzono na silniku JavaScript Engine (JSE) od Mozilli Firefox. Zbadano dwie wersje JSE – numer 1.5 oraz 2.0, dla których pozyskano źródło danych podatności *Vulnerability Database for Firefox*. Do wygenerowania grafów zależności MDG i CDG autorzy skorzystali z rezultatów analizy wykonanej przez komercyjne rozwiązania Doxygen[10] oraz Resource Standard Metrics (RSM)[11]. Wartości z utworzonego w kolejnym kroku zbioru atrybutów

[9] RMI (*Remote Method Invocation*) - RMI jest to mechanizm pozwalający na zdalne wywołanie metod obiektów
[10] http://www.doxygen.org
[11] http://msquaredtechnologies.com

wierzchołkowych i krawędziowych wykorzystane zostały do uczenia i testowania klasyfikatora. W badaniach zastosowano kilka wybranych technik klasyfikacji. Podobnie jak w modelu opisanym w poprzednim podrozdziale skorzystano z ich implementacji w środowisku WEKA. Do sprawdzenia wiarygodności wyników zaproponowanego podejścia wykorzystano 10-krotną walidację krzyżową. Wyniki sprawdzianu dla pięciu różnych technik uczenia przedstawiono w Tabela 4.

Tabela 4. Wyniki przeprowadzonej klasyfikacji przy pomocy opisywanego modelu VPM dla silnika JSE od Mozilli Firefox w wersjach 1.5 i 2.0. Źródło: [76]

Klasyfikator	Mozilla Firefox - JSE v 1.5			Mozilla Firefox - JSE v. 2.0		
	Precyzja (ang. *Precision*)	Czułość (ang. *Recall*)	Skuteczność (ang. *Accuracy*)	Precyzja (ang. *Precision*)	Czułość (ang. *Recall*)	Skuteczność (ang. *Accuracy*)
Sieci Bayesa	55%	74%	82%	58%	62%	84%
Naiwny k. Bayesa	65%	69%	86%	56%	62%	83%
Sieci neuronowe	72%	49%	85%	81%	55%	89%
Lasy losowe	62%	65%	84%	52%	58%	82%
SMO	85%	40%	86%	56%	63%	83%
Wartości średnie	**68%**	**60%**	**85%**	**61%**	**60%**	**84%**

Z tabeli wyników można zaobserwować, iż prognoza za pomocą zaproponowanego modelu VPM wskazała około 60% komponentów z podatnościami, osiągając stosunkowo wysoką sprawność klasyfikatora (na poziomie około 85%).

2.2.3.2 Modele VPM oparte o metody eksploracji tekstu

Kolejnym sposobem modelowania predykcji podatności oprogramowania jest wykorzystanie metod eksploracji tekstu. W podejściu tym zakłada się, iż kody źródłowe poszczególnych komponentów badanej wersji oprogramowania zostają w pierwszej kolejności sparsowane do postaci tzw. tokenów, które postrzegane są jako zbiór cech, docelowo wykorzystywanych do tworzenia modeli VPM. W literaturze można znaleźć co najmniej kilka prac związanych z modelowaniem predykcji podatności bazujących na metodach *text-miningu*. Krótką charakterystykę wybranych przykładów modeli z tej grupy VPM przedstawiono w kolejnych podrozdziałach.

2.2.3.2.1 Model Vulture

Przykładem modelu VPM [73], wykorzystującego metody eksploracji tekstu jest model *Vulture* [78]. W koncepcji modelu *Vulture* założono, iż w pierwszej kolejności następuje pozyskanie szczegółowych informacji dotyczących kodów źródłowych oprogramowania oraz o aktualnie zidentyfikowanych dla niego podatnościach. W tym celu należy wykonać ekstrakcję określonych danych z baz danych podatności (np. *NVD, Bugzilla,* itp.), ekstrakcję danych z repozytoriów wersji oprogramowania (np. *Common Platform*

Enumeration), oraz z kodów źródłowych poszczególnych jego komponentów. Do identyfikacji składowych poszczególnych komponentów oprogramowania autorzy modelu założyli odczyt zaimportowanych bibliotek (np. biblioteki opatrzone nagłówkiem #*include* w języku C/C++, nagłówkiem *import* w Java, itp.). Komponenty, które nie wykorzystują żadnych bibliotek lub same nie są przez żaden inny komponent importowane, są w zaproponowanym podejściu ignorowane. Analizując przykład oprogramowania zbudowanego z łącznie m komponentów, dla których zaimportowano łącznie n bibliotek, komponent zdefiniowany zostaje jako wektor zawierający n składowych, co można zapisać jako:

$$x_k = (x_{k_1}, x_{k_2}, \ldots, x_{k_n}), \text{ gdzie } x_{k_j} = \begin{cases} 1 & jeśli\ i-ty\ komponent\ importuje\ j-tą\ bibliotekę \\ 0 & w\ p.\ p \end{cases} \quad (2.15)$$

W kolejnym kroku wykonywane jest mapowanie zidentyfikowanych podatności do poszczególnej wersji komponentu oprogramowania, w którym podatność ta w przeszłości wystąpiła. W rezultacie powstaje tzw. *macierz importów* $X = (x_1, x_2, \ldots, x_m)^t$, oraz wektor podatności $Y = (y_1, y_2, \ldots, y_m)$, gdzie y_j to liczba podatności dla komponentu j. Zakładając, że wraz z rozwojem oprogramowania utworzony zostanie nowy komponent x_{m+1}, odpowiedź na pytanie czy będzie on zawierał podatności jest jednoznaczna z odpowiedzią na pytanie czy wartość $y_{m+1} > 0$. W celu uzyskania odpowiedzi autorzy zaproponowali podejście oparte na uczeniu maszynowym, w którym poprzez uczenie modelu f danymi X i y, możliwe jest dokonanie prognozy $\hat{y} = f(X)$.

Eksperyment z wykorzystaniem modelu Vulture przeprowadzono na kodach źródłowych oprogramowania przeglądarki Mozilla Firefox. Do realizacji zadań klasyfikacji autorzy skorzystali z metody wektorów nośnych (ang. *Support Vector Machines - SVMs*), zaimplementowanej w bibliotekach w języku R. Dla uwzględnionych 40 zbiorów walidacyjnych pochodzących ze zbioru Mozilla Foundation Security Advisories (*MFSA*) uzyskano precyzję klasyfikatora na poziomie około 45% i czułość wynoszącą około 65%.

2.2.3.2.2 Model autorstwa Hovsepyan, Scandariato, Walden, Joosen

Kolejny model VPM wykorzystujący metody eksploracji tekstu kodu źródłowego został zaproponowany w pracy [79]. Głównym celem badania było utworzenie modelu predykcyjnego w postaci klasyfikatora binarnego, służącego do prognozowania komponentów badanego oprogramowania, a co istotne, nie konkretnych linii kodu, u których

istnieje możliwość wystąpienia podatności na cyberzagrożenia. W odróżnieniu od prac nad innymi modelami, z uwagi na ówczesny brak ogólnodostępnych danych o podatnościach na platformę Android, zdecydowano pozyskać je za pomocą narzędzia wykonującego analizę statyczną kodu - HP FortifySCA[12]. Stąd przyjęto, iż komponent badanego oprogramowania może być uznany za podatny jeśli wspomniane narzędzie wykryje w nim obecność co najmniej jednego tzw „warningu”.

$$komponent\_podatny = \begin{cases} 1 & liczba\ "warningów"\ z\ narzędzia\ do\ analizy\ statycznej\ kodu > 0 \\ 0 & w\ p.\ p \end{cases} \quad (2.16)$$

Autorzy, wykorzystując na potrzeby eksperymentów 20 opensource’owych projektów zrealizowanych na platformę Android założyli, iż każdy komponent w zaproponowanym podejściu reprezentowany jest przez plik zawierający kod w języku programowania Java. Każdy komponent oprogramowania ponadto, potraktowano jako zbiór termów wyodrębnionych jako „tokeny” z kodu źródłowego, wraz z częstością ich wystąpienia jak przedstawiono za pomocą przykładu na Rys. 27.

HelloWorld.java	Wektor termów dla HelloWorld.java

`/* The HelloWorld class prints “Hello World!” */`
`class HelloWorld`
`{`
`  public static void main(String[] args)`
`  {`
`  System.out.println("Hello World!");`
`  }`
`}`
	args: 1, class: 2, Hello: 2, HelloWorld: 2, main: 1, out: 1, println: 1, prints: 1, public: 1, static: 1, String: 1, System: 1, The: 1, void: 1, World: 2

Rys. 27. Przykład podziału klasy HelloWorld.java na termy. Źródło: Opracowanie własne.

Przyjęto zatem ogólną postać modelu jako $komponent\_podatny = f(częstość\ wystąpień\ termów)$. Utworzony w ten sposób wektor cech (ang. *feature vector*), wraz z częstością ich wystąpień dla badanego komponentu poddany został metodom uczenia maszynowego. Powstałe modele dla dwóch technik: naiwny klasyfikator Bayesa oraz lasy losowe bazowały na pierwszych wersjach każdej z badanych aplikacji. Na potrzeby eksperymentów wykorzystano biblioteki środowiska WEKA, a za pomocą oferowanych w nim narzędzi możliwe było przeprowadzenie 10-krotnej walidacji krzyżowej dla każdej próby. Uśrednione wartości wybranych miar jakości pozwalających na ocenę klasyfikatora tj.: precyzja, czułość z przeprowadzonej analizy przedstawiono w Tabela 5.

[12] https://software.microfocus.com/en-us/products/static-code-analysis-sast/overview

Tabela 5. Wyniki badań przeprowadzonych w oparciu o opisywany model VPM, dla dwudziestu aplikacji na platformę Android. Źródło: [79]

Nazwa aplikacji na platformę Android	Naiwny klasyfikator Bayesa		Lasy losowe (ang. *Random Forest*)	
	Precyzja (ang. *Precision*)	Czułość (ang. *Recall*)	Precyzja (ang. *Precision*)	Czułość (ang. *Recall*)
AnkiDroid	85%	82%	91%	80%
BoardGameGeek	51%	32%	87%	24%
ConnectBot	99%	92%	95%	98%
CoolReader	100%	77%	89%	85%
Crosswords	90%	76%	95%	84%
FBReader	66%	74%	89%	78%
K9Mail	87%	73%	85%	91%
KeePassAndroid	85%	69%	100%	86%
MileageTracker	59%	43%	59%	43%
Mustard	83%	86%	81%	93%
Browser	88%	57%	90%	59%
Calendar	77%	75%	79%	81%
Camera	72%	49%	70%	59%
Contacts	76%	70%	75%	81%
DeskClock	79%	68%	81%	0,,78
Dialer	72%	70%	98%	100%
Email	83%	71%	83%	91%
Gallery2	70%	64%	92%	87%
Mms	80%	73%	93%	91%
QuickSearchBox	62%	80%	96%	79%

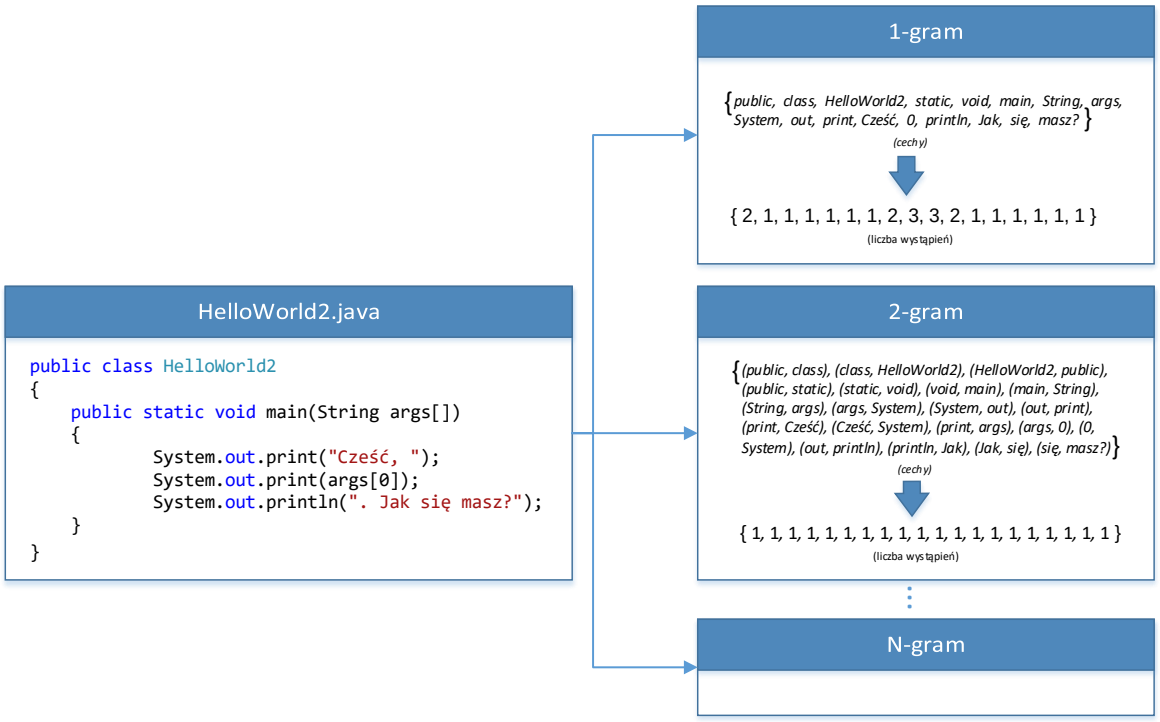
Otrzymane wyniki z eksperymentu, wynoszące odpowiednio średnio precyzja – 85%, czułość – 87% [73] [79] pozwalają stwierdzić, że zaprezentowane podejście przy użyciu techniki *Random Forest* jest obiecujące i może służyć do modelowania predykcyjnego podatności dla przyszłych wersji oprogramowania, przynajmniej z grupy oprogramowania dedykowanego na platformę Android.

2.2.3.2.3 Model VPM bazujący na n-gramach

Model VPM przedstawiony w pracy [80] to rozwiązanie, w którego założeniach zauważyć można pewne podobieństwa do modelu [79], opisanego w poprzednim podrozdziale. Autorzy niniejszego podejścia zastosowali technikę *text-mining* wyodrębniającą sekwencje tokenów z kodu źródłowego badanego oprogramowania reprezentowane w postaci n-gramów. Mianem n-gramu określa się sekwencję następujących po sobie jednostek, takich jak m.in. słowa, litery, sylaby [84]. N-gramy opierają się na statystykach i służą do przewidywania kolejnego elementu sekwencji [85]. W zależności od liczby elementów stosuję się również nazwy unigram (dla n-gramów jednoelementowych), bigram lub digram (dla sekwencji dwu elementów) i trigram (odnoszący się do ciągu trzech elementów).

W kontekście opisywanego modelu VPM, analizując kod źródłowy języka obiektowego, np. Java przyjęto, że wyodrębnione tokeny mogą być klasami, obiektami, metodami, argumentami oraz zmiennymi, a dla ułatwienia realizacji badania celowo

pominięto obecność w kodzie operatorów i separatorów. Idea wyodrębniania listy tokenów z wykorzystaniem n-gramów na przykładzie pokazowej klasy *HelloWorld2* została przedstawiona na Rys. 28.



Rys. 28. Przykład reprezentacji wyodrębnionych tokenów z klasy HelloWorld2.java przy pomocy metody N-Gram
Źródło: Opracowanie własne na podstawie [80]

Przy zastosowaniu n-gramów (dla n>1), podczas badania rozbudowanych aplikacji lub systemów, wyodrębnione wektory cech mogą charakteryzować się bardzo dużą wymiarowością. W takim przypadku może dojść do pogorszenia jakości danych uczących, co w konsekwencji może mieć wpływ na jakość utworzonych modeli. Autorzy wychodząc naprzeciw temu problemowi skorzystali z jednej z metod selekcji cech – metody rankingowej opartej na teście sumy rang Wilcoxona. Jej głównym celem była redukcja wymiaru wektora wejściowego poprzez znalezienie podzbioru cech (tokenów) opisujących komponenty badanego oprogramowania w możliwie najlepszy sposób i przez to zapewniających najwyższą jakość modelu (klasyfikatora).

Po wstępnej analizie sześciu technik uczenia maszynowego (lasy losowe, regresja logistyczna, sieci neuronowe, drzewa decyzyjne, k-najbliższych sąsiadów, Support Vector Machine - SVM) zaobserwowano najlepsze wyniki dla algorytmu SVM, który wybrano do dalszych badań. Do przeprowadzenia eksperymentów wybrano cztery aplikacje na platformę

Android z repozytorium aplikacji F-Droid[13] (*BoardGameGeek, Connectbot, CoolReader, AnkiDroid*), a dane o podatnościach poszczególnych komponentów pobrano z portalu[14] utworzonego w wyniku badań opisanych w poprzednim podrozdziale [79] [80]. W celu realizacji procesu ekstrakcji danych do postaci n-gramów ($1 \leq n \leq 5$) utworzono autorskie środowisko w środowisku Java. Proces selekcji cech zrealizowany został za pomocą skryptów w języku R, w wyniku którego pozostało do eksperymentów około 1/5 cech z pierwotnego zbioru. Następnie dla każdej aplikacji wykonano dziesięć razy 5-krotną walidację krzyżową wybierając 1/5 danych jako uczące, a pozostałą część jako zbiory testowe. Wykorzystując w eksperymentalnym środowisku funkcje z gotowej biblioteki LIBSVM[15] otrzymano potencjalnie podatne komponenty każdej z badanych aplikacji. W oparciu o przeprowadzone eksperymenty autorzy oszacowali wartości miar jakości modelu tj. precyzja i czułość, które przedstawiono w Tabela 6.

Tabela 6. Wyniki badań modelu VPM wykorzystującego n-gramy dla czterech aplikacji na platformę Android. Źródło: [80]

Aplikacja na platformę Android	Precyzja (ang. *Precision*)	Czułość (ang. *Recall*)
BoardGameGeek	97,95%	90,26%
Connectbot	95,93%	86,25%
CoolReader	97,02%	83,33%
AnkiDroid	92,22%	89%
WARTOŚCI ŚREDNIE	**95,78%**	**87,21%**

Dokonując porównania wyników otrzymanych w podejściu z pracy [79], można stwierdzić, iż zaprezentowane rozwiązanie, wykorzystujące n-gramy, wykazało wyższą skuteczność prognozy podatności czterech zbadanych aplikacji na platformę Android.

2.2.4 Pojęcie gęstości podatności

Oszacowane liczby potencjalnych podatności m.in. za pomocą modeli wykrywania podatności VDM mogą być wykorzystane do wyznaczenia miary gęstości podatności.

Def. 2.1

Gęstość podatności (ang. vulnerability density) jest to metryka służąca do oceny bezpieczeństwa oprogramowania, Określa się ją jako stosunek liczby wszystkich podatności (V) do rozmiaru oprogramowania (S), wyrażonego za pomocą standardowej miary oprogramowania, np. w liniach kodu, liczbie klas, funkcji [66]:

[13] https://f-droid.org/
[14] https://sites.google.com/site/textminingandroid/
[15] https://www.csie.ntu.edu.tw/~cjlin/libsvm/

$$V_D = \frac{V}{S} \tag{2.17}$$

W celu oszacowania liczebności zbioru wszystkich podatności w systemie istotne jest wprowadzenie pojęcia gęstości znanych podatności (ang. *known vulnerability density*) oraz pojęcia pozostałych (niewykrytych) podatności (ang. *residual vulnerability density*). Analogicznie do definicji gęstości podatności, gęstość znanych podatności (V_{KD}) wyrażana jest wzorem [66]:

$$V_{KD} = \frac{V_K}{S} \tag{2.18}$$

przy czym V_K to liczba znanych podatności dla danego oprogramowania. Gęstość pozostałych podatności w oprogramowaniu (V_{RD}) definiujemy poniższym wzorem [66]:

$$V_{RD} = V_D - V_{KD} \tag{2.19}$$

W celu oszacowania liczby wszystkich podatności, można wykorzystać modele VDM scharakteryzowane w poprzednich podrozdziałach. Dzięki takiemu podejściu, z powyższego równania uzyskana zostanie liczba podatności, które w danej chwili nie są jeszcze znane. Liczba ta może być, jednym z głównych wskaźników poziomu zagrożenia potencjalnego ataku. W praktycznym zastosowaniu metryka ta może być wykorzystana np. w procesie planowania zakupu nowej wersji danego oprogramowania, dla którego oszacowana wartość gęstości podatności powinna być porównywalna z gęstością podatności poprzednika, pod warunkiem, że zostało ono wyprodukowane przy użyciu analogicznego procesu wytwórczego.

2.3 Podsumowanie

Z przeprowadzonej analizy modeli i metod badania podatności uzyskano wiele cennych wniosków i koncepcji, które służą jako fundament w rozumieniu oraz rozwiązywaniu istotnych problemów związanych z występowaniem podatności w oprogramowaniu. Precyzyjnie zdefiniowany model prognozowania podatności oprogramowania odzwierciedlający czynniki zewnętrzne jak i wewnętrzne działające na oprogramowanie, determinujące jego podatności stanowi użyteczną wiedzę pozwalającą szacować bezpieczeństwo oprogramowania, ale również może być dobrym narzędziem dla użytkowników jak i twórców oprogramowania, służącym do prognozowania trendów wykrywania podatności oprogramowania, jak i planowania szeroko rozumianego procesu zarządzania bezpieczeństwem.

Współczesne organizacje zajmujące się tworzeniem oprogramowania stają w obliczu nieustannego wyzwania związanego z zapewnieniem bezpieczeństwa i odporności swoich produktów na potencjalne zagrożenia. Procesy zarządzania bezpieczeństwem wytwarzanego oprogramowania stanowią integralną część strategii działania tych organizacji, które koncentrują się nie tylko na samym procesie wytwarzania, lecz także na identyfikowaniu, monitorowaniu i eliminowaniu podatności, które mogą pojawić się w finalnym produkcie. Poprzez wykorzystanie ilościowych metod badania podatności scharakteryzowanych w poprzednich podrozdziałach można uzyskać istotne z punktu widzenia bezpieczeństwa informacji wymierne korzyści. Dzięki zastosowaniu dedykowanych skali oraz mechanizmów porównawczych podatności możliwe jest w sposób przybliżony określenie przy pomocy których podatności atakujący może podjąć próbę, by skompromitować analizowany system. Ponadto pozwalają one ustalić priorytety, pozwalające określić które podatności należy wyeliminować w pierwszej kolejności, czego efektem może być zminimalizowanie ryzyka potencjalnego ataku lub ograniczenie jego skutków. Analizując modele wykrywania podatności VDM, można zaobserwować, iż nawiązują one budową oraz charakterystyką działania do powszechnie wykorzystywanych modeli wzrostu niezawodności oprogramowania i służą do oceny profilu bezpieczeństwa oprogramowania jako całości, poprzez prognozowanie liczby podatności oprogramowania. Konstrukcja większości modeli predykcji podatności VPM natomiast, oparta na technikach uczenia maszynowego, umożliwia dokonanie prognozy, której wynikiem będzie wskazanie potencjalnego wystąpienia lub braku podatności w poszczególnych składowych oprogramowania (komponentach, klasach, plikach, itd.). W praktycznym zastosowaniu modele VDM oraz VPM mogą służyć do wyznaczania kierunków właściwej alokacji zasobów do testowania, aktualizacji i wprowadzania łatek oprogramowania.

W opisywanych podejściach niejednokrotnie bazowano m.in. na danych historycznych o wystąpieniu podatności w badanym oprogramowaniu czy jego atrybutach, mających niejednokrotnie wpływ na wystąpienie w nim podatności w przyszłości. Z uwagi na fakt, iż źródeł wiedzy o podatnościach dla oprogramowania ogólnodostępnego może być bardzo wiele, trudno jest uzyskać precyzyjne wyniki prognozy, ponieważ nie każdy testujący może być zainteresowany udzieleniem informacji o wykrytej podatności. W niniejszej pracy ograniczono się zatem do rodzaju oprogramowania, dla którego z uwagi na ograniczoną dostępność, liczba potencjalnych źródeł informacji o podatnościach może być mniejsza niż dla oprogramowania ogólnodostępnego dla użytkowników.

Ponadto, mając na uwadze, że zgodnie z istniejącymi trendami, zarządzanie podatnościami oprogramowania staje się kluczowym elementem całego cyklu życia tworzenia oprogramowania, w pracy uwzględniono wykorzystanie zgromadzonych informacji o czasie oraz intensywności występowania podatności w wytwarzanym oprogramowaniu, co może mieć istotny wpływ na doskonalenie procesów zarządzania bezpieczeństwem. Jednocześnie, w przeciwieństwie do w większości używanych współcześnie modeli prognozowania podatności, w niniejszych rozważaniach uwzględniono aspekt stosowania działań profilaktyczno-naprawczych, mających na celu eliminowanie podatności. Szczegółową charakterystykę autorskich modeli prognozowania podatności oprogramowania, wykorzystujących własności łańcuchów Markowa oraz zastosowanej nowatorskiej metodologii w kontekście analizy problematyki prognozowania występowania podatności w oprogramowaniu przedstawiono w następnym rozdziale.

3. Autorskie modele prognozowania podatności oprogramowania

3.1 Elementy probabilistyki i statystyki wykorzystane w pracy

W niniejszym podrozdziale dokonano wprowadzenia kluczowych pojęć i narzędzi z obszaru probabilistyki i statystyki, takie jak:

- łańcuchy Markowa,
- prawdopodobieństwo dojścia i średni czas dojścia,
- rozkład graniczny i stacjonarny.

Opracowano je na podstawie następujących pozycji z literatury: [86] [87] [88] [89] oraz [90] i stanowią fundament budowy autorskich modeli i metod prognozowania podatności oprogramowania, przedstawionych w dalszej części pracy. Aby zapewnić ich pełniejsze zrozumienie w Załączniku 1 do niniejszej pracy zawarto opisy podstawowych definicji i właściwości z dziedziny probabilistyki, które stanowią niezbędne tło dla dalej opisanych metod.

3.1.1 Łańcuchy Markowa

Podrozdział ten został opracowany na podstawie danych zaczerpniętych z rozdziału 1.1, zawartego w źródle [88] oraz z literatury oznaczonej jako [90].

Niech I będzie zbiorem przeliczalnym (tzn. zbiorem równolicznym z $\mathbb{N}$). Każdy element $i \in I$ nazwiemy **stanem**, natomiast zbiór I, **przestrzenią stanów**. Z definicji miary mamy, że $\lambda = (\lambda_i : i \in I)$ jest miarą na I jeśli $0 \leq \lambda_i < \infty$ dla każdego $i \in I$, z definicji rozkładu prawdopodobieństwa mamy, że λ jest rozkładem jeśli $\sum_{i \in I} \lambda_i = 1$. Ustalmy przestrzeń probabilistyczną $(\Omega, \mathcal{F}, \mathbb{P})$ i niech $X: \Omega \to I$ będzie zmienną losową. Niech teraz

$$\lambda_i = \mathbb{P}(X = i) = \mathbb{P}(\{\omega : X(\omega) = i\}). \tag{3.1}$$

Wtedy λ definiuje rozkład zmiennej losowej X.

Powiemy, że macierz $P = (p_{ij} : i, j \in I)$ jest **stochastyczna**, jeśli każdy wiersz $(p_{ij} : j \in I)$ sumuje się do jedności.

Powiemy, że $(X_n)_{n \geq 0}$ jest **łańcuchem Markowa** z początkowym rozkładem λ i macierzą przejścia P jeśli dla $n \geq 0$ oraz $i_0, \dots, i_{n+1} \in I$

- $\mathbb{P}(X_0 = i_0) = \lambda_{i_0}$;
- $\mathbb{P}(X_{n+1} = i_{n+1} | X_0 = i_0, \dots, X_n = i_n) = \mathbb{P}(X_{n+1} = i_{n+1} | X_n = i_n) \coloneqq p_{i_n i_{n+1}}$. (3.2)

Wówczas powiemy, w skrócie, że $(X_n)_{n\geq 0}$ jest typu $\mathbf{Markov}(\boldsymbol{\lambda}, \boldsymbol{P})$.

Jeśli dany wiersz macierzy przejścia składa się z jedynki i zera, wówczas stan łańcucha Markowa odpowiadający temu wierszowi nazwiemy **stanem absorbującym**.

Przytoczymy teraz dwa fundamentalne twierdzenia teorii łańcuchów Markowa. Pierwsze jest warunkiem koniecznym i dostatecznym na to, aby proces był łańcuchem Markowa. Poniższe twierdzenia w znaczący sposób uproszcza obliczenia na łańcuchach Markowa [88].

Twierdzenie 1

Proces losowy o dyskretnym czasie $(X_n)_{0\leq n\leq N}$ $(X_n: \Omega \to I)$ *jest typu* $\mathrm{Markow}(\lambda, P)$ *wtedy i tylko wtedy gdy dla każdych stanów* $i_0, \dots, i_N \in I$

$$\mathbb{P}(X_0 = i_0, X_1 = i_1, \dots, X_N = i_N) = \lambda_{i_0} p_{i_0 i_1} p_{i_1 i_2} p_{i_2 i_3} \cdots p_{i_{N-1} i_N}. \tag{3.3}$$

Kolejne twierdzenie pokazuje, sposób w jaki możemy obliczyć prawdopodobieństwo przejścia w n krokach od stanu i do stanu j. Z poniższego twierdzenia będziemy w bezpośredni sposób korzystali w dalszej części pracy [88].

Twierdzenie 2

Niech proces $(X_n)_{n\geq 0}$ *będzie typu* $\mathrm{Markow}(\lambda, P)$*, wtedy dla każdych* $m, n \geq 0$ *zachodzą następujące warunki,*

- $\mathbb{P}(X_n = j) = \sum_{i\in I} \lambda_i p_{ij}^{(n)}$,
- $\mathbb{P}(X_{n+m} = j | X_m = i) = p_{ij}^{(n)}$, (3.4)

gdzie $p_{ij}^{(n)}$ *jest elementem znajdującym się w* i*-tym wierszu i* j*-tej kolumnie macierzy* P^n.

3.1.2 Prawdopodobieństwo dojścia i średni czas dojścia

Niech $(X_n)_{n\geq 0}$ będzie typu $\mathrm{Markow}(\lambda, P)$ oraz niech A będzie podzbiorem przestrzeni stanów I. Czasem dojścia do podzbioru A przestrzeni stanów I nazywamy zmienną losową $H^A: \Omega \to \{0,1,2, \dots\} \cup \{\infty\}$ zdefiniowaną następującym wzorem

$$H^A(\omega) = \inf\{n \geq 0 : X_n(\omega) \in A\}. \tag{3.5}$$

Będziemy przyjmowali konwencję, że $\inf \emptyset = \infty$. Prawdopodobieństwo, że proces $(X_n)_{n\geq 0}$ wychodzący ze stanu i dojdzie do zbioru A jest wówczas określone następująco

$$h_i^A = \mathbb{P}\left(X_{H^A} \in A \middle| X_0 = i\right). \tag{3.6}$$

Jeśli A jest zamkniętą klasą (powiemy, że zbiór A jest klasą zamkniętą jeśli $\mathbb{P}(X_n = j \text{ dla pewnego } n \geq 0) > 0$, to $j \in A$), to h_i^A nazywamy **prawdopodobieństwem dojścia** (ang. *hitting probability*).

Średni czas dojścia (ang. *mean hitting time*) procesu $(X_n)_{n\geq 0}$ do stanów zbioru A, wychodząc ze stanu i będziemy oznaczali przez k_i^A.

Okazuje się, że obliczenie wartości k_i^A oraz h_i^A sprowadza się do rozwiązania odpowiednich układów równań liniowych związanych z macierzą przejścia P. Innymi słowy, mając dana macierz przejścia P możemy obliczyć szukane wartości. Fakty te zestawimy w dwóch poniższych twierdzeniach.

Twierdzenie 3 *Wektor prawdopodobieństwa dojścia* $\boldsymbol{h^A = (h_i^A : i \in I)}$ *jest minimalnym niezerowym rozwiązaniem układu równań liniowych*

$$\begin{cases} h_i^A = 1 & \text{dla } i \in A \\ h_i^A = \sum_{j \in A} p_{ij} h_j^A & \text{dla } i \notin A. \end{cases} \tag{3.7}$$

(Przez minimalne rozwiązanie rozumiemy, że jeśli $x = (x_i : i \in I)$ *jest rozwiązaniem takim, że* $x_i \geq 0$ *dla każdego* $i \in I$*, to* $x_i \geq h_i$ *dla każdego* i*.)*

Twierdzenie 4 *Wektor średnich czasów dojścia* $\boldsymbol{k^A = (k_i^A : i \in I)}$ *jest minimalnym rozwiązaniem układu równań liniowych*

$$\begin{cases} k_i^A = 0 & dla\ i \in A \\ k_i^A = 1 + \sum_{j \notin A} p_{ij} k_j^A & dla\ i \notin A. \end{cases} \tag{3.8}$$

Uwaga. Zauważmy, że układy równań z Twierdzeń 3 i 4 mają jednoznaczne rozwiązanie jeśli macierz przejścia jest nieosobliwa. Dodatkowo jeśli łańcuch Markowa nie posiada stanu absorbującego rozwianie układu z Twierdzenia 3 składa się z samych jedynek. Natomiast wektor z Twierdzenia 4, w tym wypadku będzie składał się z nieskończoności.

3.1.3 Rozkład graniczny i stacjonarny

Powiemy, ze łańcuch Markowa $(X_n)_{n\in\mathbb{N}}$ ma **rozkład graniczny** jeśli dla każdego $i.j \in I$ istnieje granica

$$\lim_{n\to\infty} \mathbb{P}(X_n = j|X_0 = i) \tag{3.9}$$

oraz powyższe granice tworzą rozkład prawdopodobieństwa na przestrzeni stanów I, tzn.

$$\sum_{j\in I} \lim_{n\to\infty} \mathbb{P}(X_n = j|X_0 = i) = 1. \tag{3.10}$$

Jeśli proces $\mathrm{Markow}(\lambda, P)$ jest określony na skończonej przestrzeni stanów oraz macierz przejścia P jest **regularna** (tzn. jej kolejne potęgi nie są macierzami zerowymi), wówczas istnieje rozkład graniczny.

Powiemy, że rozkład prawdopodobieństwa $\pi = (\pi_i : i \in I)$ jest **stacjonarny** jeśli

$$\pi = \pi P. \tag{3.11}$$

Innymi słowy, rozkład jest stacjonarny jeśli jest niezmiennikiem przy prawostronnym mnożeniu przez macierz przejścia P.

Okazuje się, że w pewnych przypadkach rozkład graniczny pokrywa się z rozkładem stacjonarnym. Zanim przytoczymy odpowiednie twierdzenie, konieczne jest wprowadzenie pewnych klas stanów.

Stan $j \in I$ nazwiemy **osiągalnym** ze stanu $i \in I$ jeśli istnieje skończona liczba $n \geq 0$ taka, że

$$[P^n]_{i,j} = \mathbb{P}(X_n = j|X_0 = i) > 0. \tag{3.12}$$

Jeśli stan $i \in I$ jest osiągalny ze stanu $j \in I$ oraz stan j jest osiągalny ze stanu i, wówczas mówimy, że **stany są skomunikowane**.

Skomunikowanie jest relacją równoważności (tzn. relacją zwrotną, symetryczną i przechodnią) na przestrzeni stanów I. Klasy abstrakcji tej relacji nazwiemy **klasami skomunikowania**.

Łańcuch Markowa nazwiemy **nierozkładalnym** jeśli na przestrzeni stanów istnieje dokładnie jedna klasa skomunikowania.

Niech dany będzie stan $i \in I$, rozważmy następujący zbiór

$$\{n \in \mathbb{N}: n \geq 1, [P^n]_{i,i} > 0\}. \tag{3.13}$$

Okresem stanu i nazwiemy największy wspólny dzielnik powyższego zbioru. Stan posiadający okres równy 1, nazwiemy **aperiodycznym**. Łańcuch Markowa jest aperiodyczny jeśli wszystkie jego stany są aperiodyczne.

Twierdzenie 5

Załóżmy, że łańcuch Markowa $(X_n)_{n\in\mathbb{N}}$ *jest nierozkładalny i aperiodyczny. Wówczas łańcuch* $(X_n)_{n\in\mathbb{N}}$ *posiada rozkład graniczny*

$$\pi_i := \lim_{n\to\infty} \mathbb{P}(X_n = i | X_0 = j) = \lim_{n\to\infty} [P^n]_{j,i}, \quad i,j \in I, \tag{3.14}$$

który jest również rozkładem stacjonarnym wyznaczonym jednoznacznie przez równanie (3.11)

Uwaga. Jeśli łańcuch Markowa posiada stan absorbujący wówczas rozkład graniczny tego stanu będzie stale równy jeden natomiast pozostałych będzie wynosił stale 0.

3.2 Obszar wykorzystania zaproponowanych modeli i metod

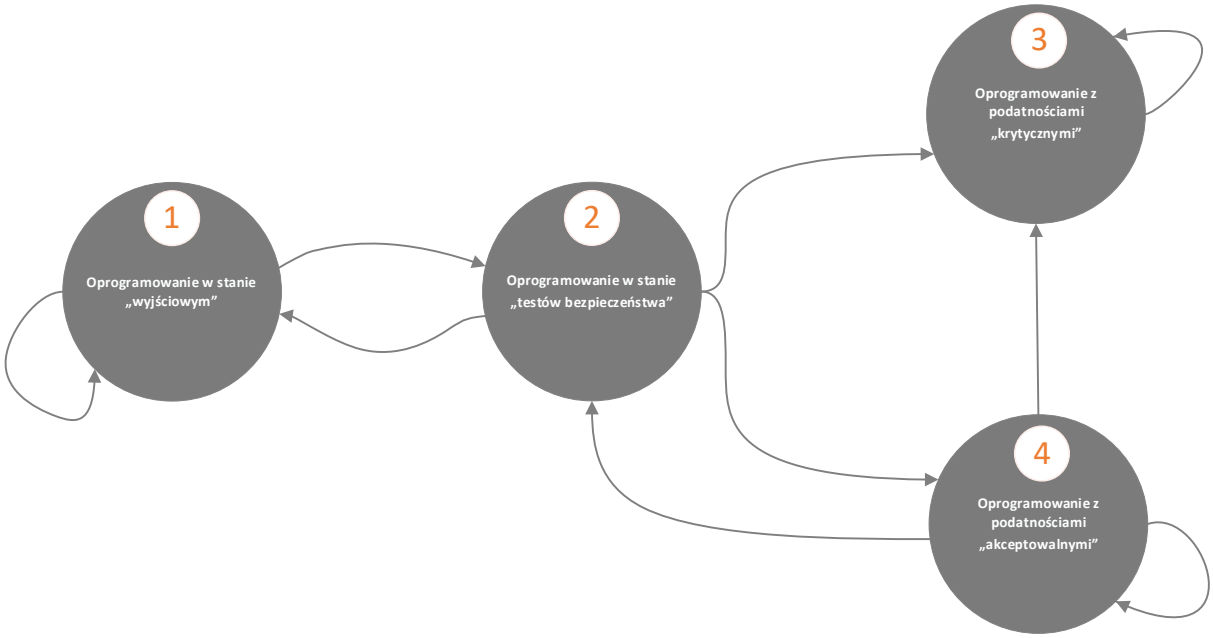
Proces wykrywania podatności oprogramowania w ujęciu ogólnym może być rozpatrywany na wielu płaszczyznach. Jednym z pierwszych etapów tego procesu jest zazwyczaj wykonanie szeroko rozumianych testów bezpieczeństwa oprogramowania przed jego wypuszczeniem do użycia, które pozwalają uzyskać wstępną informację o potencjalnych podatnościach w badanym oprogramowaniu. W przypadku, gdy analizujemy oprogramowanie, które jest ogólnodostępne na rynku dla użytkowników, informacje o jego podatnościach mogą jednak również pochodzić z innych źródeł, m.in. właśnie od użytkowników, z badań niezależnych podmiotów, z raportów po zaistniałych incydentach i wielu innych. Z powyższego wynika, iż bardzo utrudnione jest zadanie polegające na pozyskaniu rzetelnych danych do przeprowadzenia kompleksowego procesu prognozowania podatności dla takiego ogólnodostępnego oprogramowania (np. pakiety biurowe, oprogramowania antywirusowe, systemy operacyjne, itd.), z uwzględnieniem wszystkich źródeł informacji o podatnościach.

Organizacje starające się sprostać wymaganiom rynku coraz częściej kładą nacisk na weryfikację bezpieczeństwa swoich aplikacji poprzez przeprowadzanie testów, które nie tylko ujawniają istniejące podatności, ale również dostarczają istotnych wniosków wywodzących się z analizy tychże testów. W niniejszej pracy ograniczono się zatem do zamodelowania procesu wykrywania podatności dla oprogramowania dedykowanego, stale

rozwijanego, dla którego zakłada się, że zebrane informacje o podatnościach są wynikiem w dużej mierze przeprowadzenia szerokiej gamy testów bezpieczeństwa. Za przykład oprogramowania dedykowanego, możemy podać takie, którego projekt i implementacja zostały zlecone przez dany podmiot innemu podmiotowi i nie jest przewidziana jego dystrybucja dla odbiorców z zewnątrz – między innymi można tu wymienić systemy czy aplikacje tworzone dla instytucji kluczowych z punktu widzenia bezpieczeństwa i funkcjonowania państwa. W przypadku oprogramowania dedykowanego liczba potencjalnych testujących, w tym atakujących, staje się ograniczona. Automatycznie otrzymujemy wtedy zmniejszoną liczbę źródeł informacji o potencjalnych podatnościach, przez co prognozy wystąpienia podatności, oparte na uzyskanych informacjach z testów bezpieczeństwa oprogramowania mogą okazać się bardziej precyzyjne.

3.3 Model ogólny - bez uwzględnienia działań profilaktyczno-naprawczych

Ogólny model wykrywania podatności oprogramowania będzie bazował na autorskim cztero-stanowym Łańcuchu Markowa przedstawionym na schemacie Rys. 29.



Rys. 29 Ogólny model wykrywania podatności

W powyższym łańcuchu stany interpretujemy w następujący sposób:

- Stan 1 – oprogramowanie w stanie tzw. „wyjściowym”, w którym zakładany jest brak informacji o występujących podatnościach, będący wynikiem jednego z kolejno wymienionych przypadków:
    - oprogramowanie nowoutworzone, przed podjęciem decyzji o wyszukiwaniu podatności;
    - w testach bezpieczeństwa nie wykryto podatności;
- Stan 2 – oprogramowanie w stanie testów bezpieczeństwa;
- Stan 3 – oprogramowanie w stanie z wykrytymi podatnościami krytycznymi (tzn. dla co najmniej jednej wykrytej podatności podczas testów bezpieczeństwa oszacowany wskaźnik zagrożenia podatności (ang. v*ulnerability severity*) przekracza subiektywnie ustalony poziom), w wyniku czego zakłada się, że oprogramowanie nie może być dopuszczone do użytkowania;
- Stan 4 – oprogramowanie z wykrytymi podatnościami tzw. „akceptowalnymi” (tzn. dla żadnej z wykrytych podatności podczas testów bezpieczeństwa oszacowany wskaźnik zagrożenia podatności (ang. v*ulnerability severity*) nie przekracza subiektywnie ustalonego poziomu, w wyniku czego zakłada się, że oprogramowanie może być dopuszczone do użytkowania).

W celu zachowania spójności nazewnictwa Stany 1-4 nazywane będą w dalszej części pracy stanami bazowymi. Należy również zaznaczyć, że oprogramowanie może zostać przekazane do użytkowania podczas gdy znajduje się w stanie 1 lub 4. Przejścia między odpowiednimi stanami bazowymi opisujemy następująco:

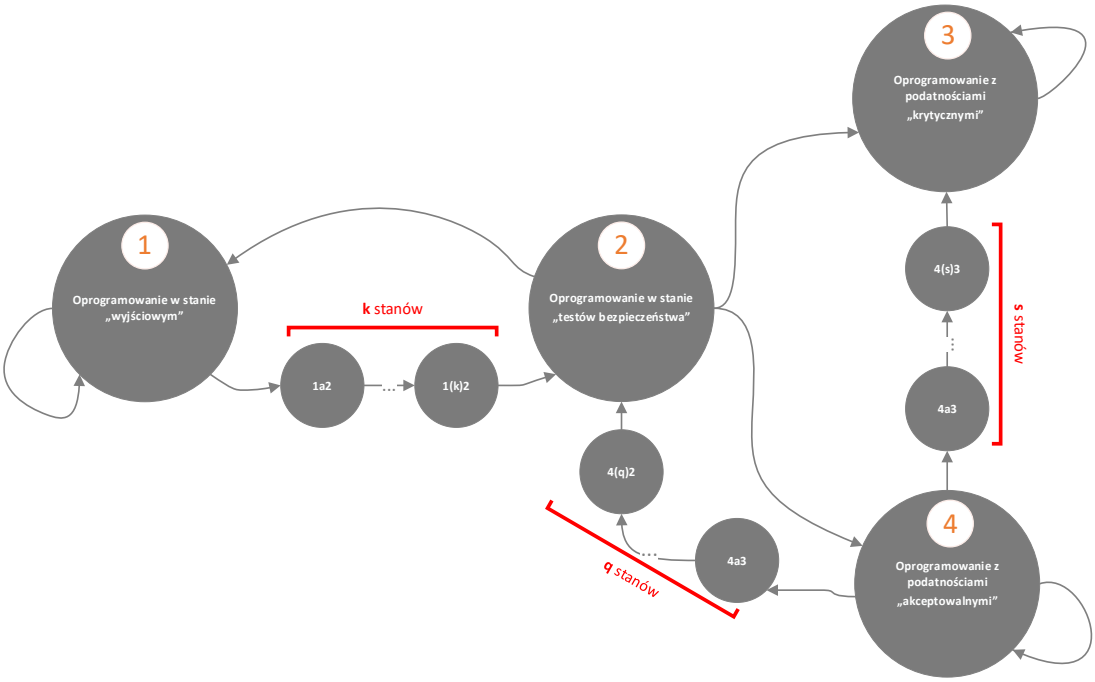
- $1 \rightarrow 1$ – przejście można zinterpretować na kilka sposobów:
    - Faza implementacji oprogramowania się zakończyła i jest ono gotowe do przetestowania, a jednocześnie (dla tego wariantu oprogramowanie nie jest wdrożone):
        - nie ma decyzji o rozpoczęciu testów bezpieczeństwa oprogramowania;
        - następuje oczekiwanie na wykonanie testów bezpieczeństwa oprogramowania;

- Po zakończonych testach bezpieczeństwa nie stwierdzono podatności i oprogramowanie zostało wdrożone lub nie ma decyzji o wdrożeniu do użycia.

- $1 \rightarrow 2$ – rozpoczęcie testów bezpieczeństwa dla danej wersji oprogramowania;
- $2 \rightarrow 1$ – nie wykryto podatności podczas testów bezpieczeństwa danej wersji oprogramowania;
- $2 \rightarrow 3$ – wykryto podatności sklasyfikowane jako krytyczne podczas testów bezpieczeństwa danej wersji oprogramowania;
- $2 \rightarrow 4$ – wykryto podatności tzw. „akceptowalne” podczas testów bezpieczeństwa danej wersji oprogramowania;
- $3 \rightarrow 3$ – brak możliwości podjęcia działań profilaktyczno-naprawczych, w celu usunięcia podatności krytycznych (np. z uwagi na brak odpowiednich zasobów);
- $4 \rightarrow 2$ – decyzja o kolejnej iteracji testów bezpieczeństwa oprogramowania, w którym wykryto podatności „akceptowalne”, m.in.:
  - decyzja podjęta arbitralnie;
  - w przypadku zastosowania zmian do oprogramowania (niezwiązanych z usuwaniem podatności), zaleca się ponowne wykonanie testów bezpieczeństwa (pozwoli to m.in. zbadać czy wykryte wcześniej podatności akceptowalne nadal występują i czy czasem nie pojawiły się nowe podatności);
- $4 \rightarrow 4$ – brak działań profilaktyczno-naprawczych w celu eliminacji podatności, przy świadomości, że subiektywnie „akceptowalne” podatności istnieją, oprogramowanie może zostać dopuszczone do środowiska produkcyjnego oraz może być nadal rozwijane;
- $4 \rightarrow 3$ – przeklasyfikowanie podatności akceptowalnych na krytyczne, np.:
  - w przypadku wykrytej podatności, którą wstępnie sklasyfikowano jako akceptowalną, jeśli nadal nie występuje w referencyjnej bazie danych, wykonano ponowie proces klasyfikacji, w oparciu o zaktualizowaną lub inną próbkę danych do klasyfikacji;

- nastąpiła zmiana parametrów konfiguracji środowiska produkcyjnego, w którym pracowało oprogramowanie, co przełożyło się na aktualizację parametrów klasyfikacji podatności.

Zakładamy, że przejścia: $1 \rightarrow 1$, $2 \rightarrow 3$, $2 \rightarrow 4$, $2 \rightarrow 1$, $3 \rightarrow 3$ oraz $4 \rightarrow 4$ są przejściami jednokrokowymi (zakładamy, że jeden krok przejścia ma wartość jednej ustalonej jednostki czasu, np. 1 miesiąc). Natomiast przejścia $1 \rightarrow 2$, $4 \rightarrow 2$ oraz $4 \rightarrow 3$ są przejściami wielokrokowymi, co należy interpretować, że dla przejść pomiędzy poszczególnymi stanami bazowymi dodano co najmniej jeden stan określany w dalszej części pracy jako stan przejściowy (zakładamy również, że pojedyncze przejście pomiędzy każdym stanem w przejściu wielokrokowym, ma wartość jednej ustalonej jednostki czasu, tej samej jak ustalono dla przejścia jednokrokowego). Liczbę kroków przejściowych będziemy szacowali na podstawie zebranych danych. Dokładniej, w pierwszej kolejności będziemy estymowali rozkłady czasu przejść między odpowiednimi stanami bazowymi. Mając rozkłady będziemy mogli znaleźć wartości oczekiwane odpowiednich przejść. Zaokrąglone wartości oczekiwane będą reprezentowały liczbę kroków w przejściu między odpowiednimi stanami bazowymi. Wówczas będziemy mogli zbudować szczegółowy model wykrywania podatności w którym sztucznie dopisujemy oszacowaną ilość stanów przejściowych. Przedstawiliśmy go na poniższym rysunku.



Rys. 30. Szczegółowy model wykrywania podatności

Rozważając proces prognozowania podatności oprogramowania w oparciu o zaproponowane łańcuchy Markowa, ich ahistoryczność, polegająca na tym, że przyszły stan zależy wyłącznie od stanu bieżącego, można uzasadnić na kilka sposobów:

- **Naturalne sekwencje zdarzeń**

W praktyce wiele procesów w rozwoju oprogramowania jest z natury sekwencyjnych, gdzie:

  - proces przechodzi przez określone etapy w sposób przewidywalny i zależny głównie od bieżącego stanu;
  - każdy etap dostarcza wystarczających informacji do podjęcia kolejnych kroków.

- **Wystarczająca dokładność predykcji**

W wielu przypadkach obecny stan rzeczywiście zawiera wystarczające informacje do prognozowania przyszłych stanów. Dla przykładu, wyniki testów bezpieczeństwa (Stan 2) bezpośrednio wpływają na wykrycie podatności krytycznych lub akceptowalnych (Stany 3 i 4). Nie ma zatem potrzeby uwzględniania poprzednich wyników testów, ponieważ bieżące wyniki są najbardziej aktualne i istotne.

- **„Lokalność" informacji**

Łańcuchy Markowa zakładają, że przyszły stan systemu zależy wyłącznie od jego obecnego stanu, a nie od pełnej historii wcześniejszych stanów. W kontekście oprogramowania można to uzasadnić następująco:

  - aktualny stan oprogramowania (np. wyniki najnowszych testów bezpieczeństwa) zawiera wystarczające informacje do podejmowania decyzji dotyczących przyszłych działań;
  - wiele decyzji w procesie rozwoju i testowania oprogramowania opiera się na bieżących wynikach i statusie, a nie na pełnej historii.

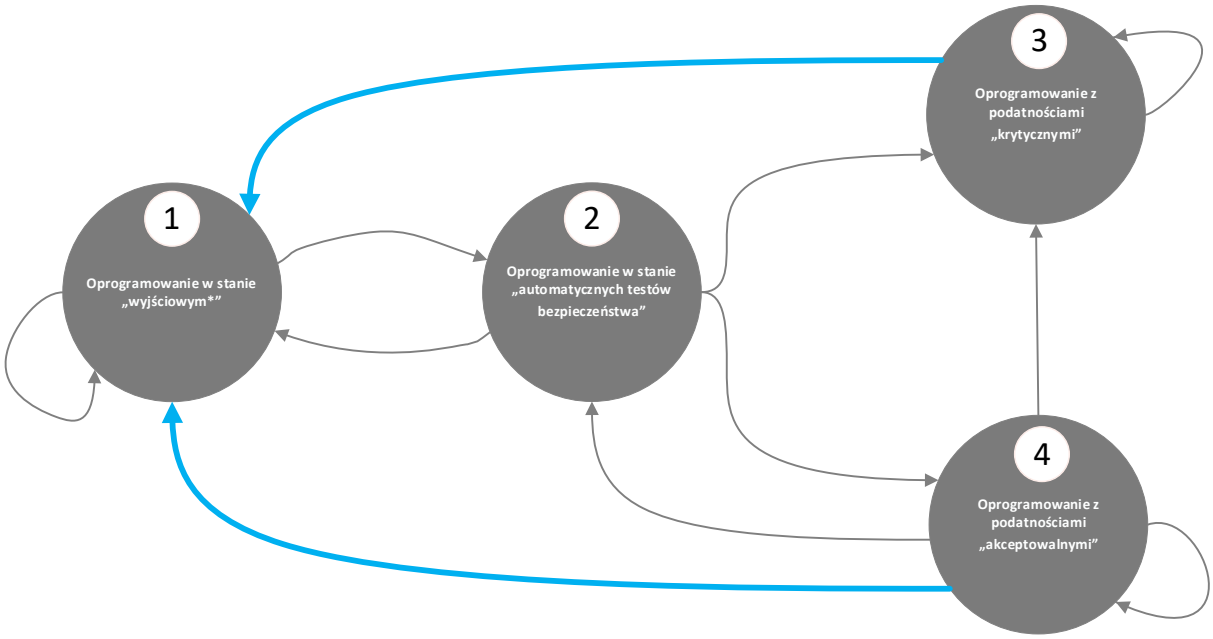
- **Redukcja złożoności**

Ahistoryczność łańcucha Markowa pozwala na znaczne uproszczenie modelu. Dzięki temu:

  - modele stają się łatwiejsze do implementacji i wymagają mniej zasobów obliczeniowych;
  - eliminowana jest konieczność przechowywania i analizy pełnej historii zdarzeń, co jest korzystne w przypadku dużych i skomplikowanych systemów.

3.4 Model ogólny - z uwzględnieniem działań profilaktyczno-naprawczych

W celu rozważenia działań profilaktyczno-naprawczych w powyższych modelach rozważmy zmodyfikowany łańcuch Markowa przedstawiony na Rys. 31.



Rys. 31. Ogólny model wykrywania podatności z uwzględnieniem działań profilaktyczno-naprawczych

W wyniku rozszerzenia modelu, należy uzupełnić interpretację Stanu 1 o jeden przypadek:

- oprogramowanie po zastosowaniu łatek bezpieczeństwa (działań profilaktyczno-naprawczych);

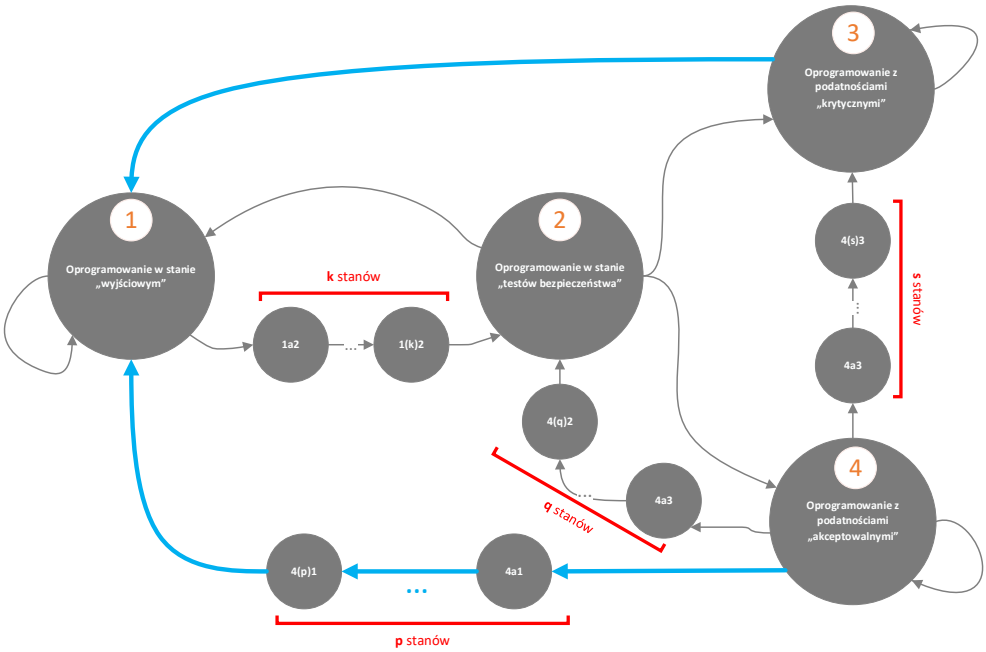
Przejścia między odpowiednimi stanami bazowymi interpretujemy w ten sam sposób jak w przypadku modelu bez uwzględnienia działań profilaktyczno-naprawczych. Dodatkowo dochodzą dwa przejścia:

- 3 → 1 – podjęto działania profilaktyczno-naprawcze w celu usunięcia podatności krytycznych (na potrzeby przyjętego modelu zakłada się, iż w przypadku wykrycia podatności krytycznych należy podjąć natychmiastowe działania w celu ich usunięcia, a w tym przypadku oprogramowanie nie może zostać dopuszczone do wdrożenia);
- 4 → 1 - podjęto decyzję o wyeliminowaniu podatności akceptowalnych.

Dodatkowo interpretację przejścia 1 → 1 należy rozszerzyć o jeden przypadek:

- W rezultacie wykonanych poprawek bezpieczeństwa, zakłada się, że oprogramowanie jest wolne od podatności i można je wdrożyć lub zostało wdrożone.

Analogicznie jak poprzednio rozważamy model szczegółowy, który uwzględnia przejścia wielokrokowe. Nowe przejście $4 \rightarrow 1$, uwzględniamy jako wielokrokowe, natomiast przejście $3 \rightarrow 1$ jako jednokrokowe. Jednocześnie, poczynione założenia w rozdziale 3.3 co do stosowalności łańcuchów Markowa pozostaje w mocy również przy uwzględnieniu działań profilaktyczno-naprawczych, ponieważ ich implementacja wynika z rezultatu wykonanych testów bezpieczeństwa w stanie 2. Na rysunku Rys. 32 przedstawiono model, w którym wybrane przejścia między stanami bazowymi są wielokrokowe.



Rys. 32. Szczegółowy model wykrywania podatności z uwzględnieniem działań profilaktyczno-naprawczych

Na podstawie tak określonych łańcuchów Markowa definiujemy następujące pojęcia:

- **Poziom bezpieczeństwa oprogramowania** jest określony za pomocą modelu bez działań profilaktyczno-naprawczych (podrozdział 3.3) oraz poniższego wzoru:

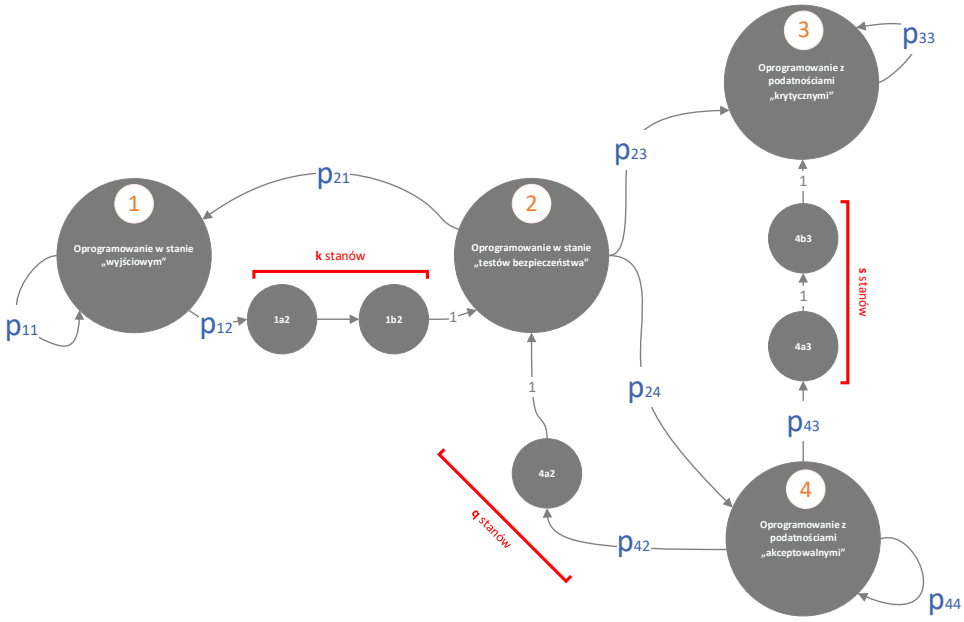
$$1 - \mathbb{P}(X_{H^4} = 4 | X_0 = 1). \qquad (3.15)$$

Tzn. prawdopodobieństwo dojścia do stanu 4 (w którym oprogramowanie ma podatności, ale dopuszczone jest do działania w środowisku produkcyjnym – uznajemy za stan niebezpieczny) traktujemy jako poziom niebezpieczeństwa, więc szacowaną wartością jest dopełnienie tego (1 – to prawdopodobieństwo dojścia do stanu 4).

- **Intensywność wykrywania krytycznych podatności oprogramowania** jest wyrażona przez k_1^3 (średni czas dojścia ze stanu 1 do stanu 3) w modelu bez działań profilaktyczno-naprawczych (podrozdział 3.3).
- **Działania profilaktyczno-naprawcze** uznajemy za skuteczne jeśli rozkład graniczny dojścia do stanu 4 jest mniejszy niż rozkład graniczny dojścia do stanu 1 ($\pi_4 < \pi_1$). W ujęciu praktycznym działania profilaktyczno-naprawcze na określonym oprogramowaniu możemy uznać za skuteczne, jeśli przy długim przebiegu łańcuchu, tzn. w wyniku wielokrotnego testowania określonego oprogramowania, wraz z wydaniem każdej jego kolejnej wersji, częściej znajdujemy się w stanie 1 niż 4. W tym wypadku korzystamy z modelu z działaniami profilaktyczno-naprawczymi.

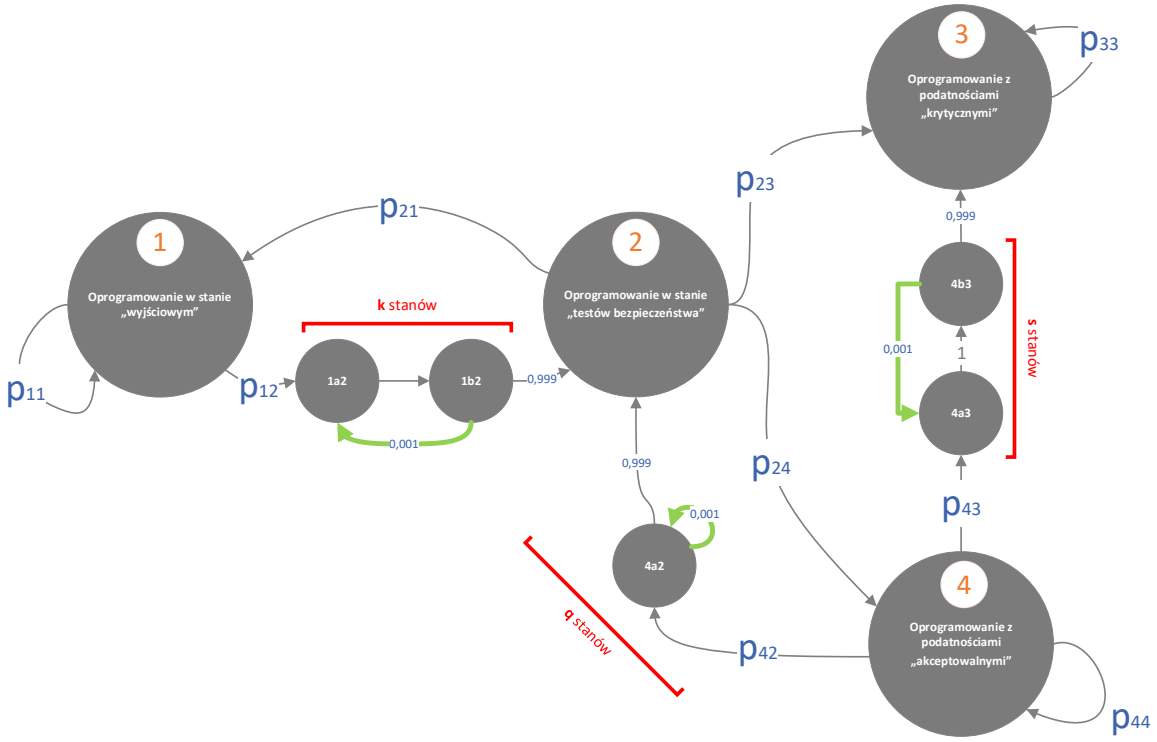
3.5 Uwagi natury obliczeniowej

W momencie, gdy dla zaproponowanych łańcuchów Markowa wprowadzamy przejścia wielokrokowe pomiędzy stanami bazowymi (rysunek Rys. 30 i Rys. 32), łatwo zauważyć, że ich macierz przejścia będzie macierzą osobliwą. Co w efekcie będzie prowadziło do niejednoznacznych rozwiązań układów z Twierdzeń 3 i 4. W celu obejścia tej subtelności konieczne jest wprowadzenie małej poprawki do modelu. Dokładniej załóżmy, że w modelu bez działań profilaktyczno-naprawczych oszacowaliśmy następujące wartości: $k = s = 2,\ q = 1$. Wówczas łańcuch Markowa ma poniższą postać:



Rys. 33. Przykładowy model bez poprawki z uwzględnieniem przejść wielokrokowych między stanami bazowymi.

Na rysunku Rys. 33 czerwonymi klamrami zaznaczono stany łańcucha Markowa, które zostały dodane w celu zbudowania przejścia wielokrokowego między stanami bazowymi. W takiej postaci prawdopodobieństwo przejścia między ostatnim stanem przejściowym, a stanem bazowym wynosi 1 (jak zaznaczono na Rys. 33), w wyniku czego macierz przejścia tego modelu stała się macierzą osobliwą. W celu ominięcia tego problemu wprowadzamy tzw. zaburzenie przejścia z ostatnich stanów przejściowych do stanów bazowych dla każdego przejścia wielokrokowego. Tzn. prawdopodobieństwo przejścia z ostatnich stanów przejściowych do stanów bazowych zmieniamy z 1 na 0,999 oraz dodajemy dodatkowe przejście z ostatnich stanów przejściowych do pierwszych stanów przejściowych w każdym przejściu wielokrokowym. Prawdopodobieństwo tego przejścia ustalamy na 0,001. Opisany powyżej sposób uniknięcia osobliwości macierzy dla przedstawionego modelu pokazano na poniższym rysunku.



Rys. 34. Przykładowy model z poprawką z uwzględnieniem przejść wielokrokowych między stanami bazowymi

Po takiej poprawce, macierz przejścia modelu jest już nieosobliwa.

Zauważmy, że w modelu z uwzględnieniem działań profilaktyczno-naprawczych, nie jesteśmy w stanie obliczyć poziomu bezpieczeństwa tak jak to było w przypadku modelu bez ich zastosowania, ponieważ w tym modelu łańcuch Markowa nie posiada stanu absorbującego. Dlatego też, skuteczność tych działań mierzymy przy użyciu rozkładu

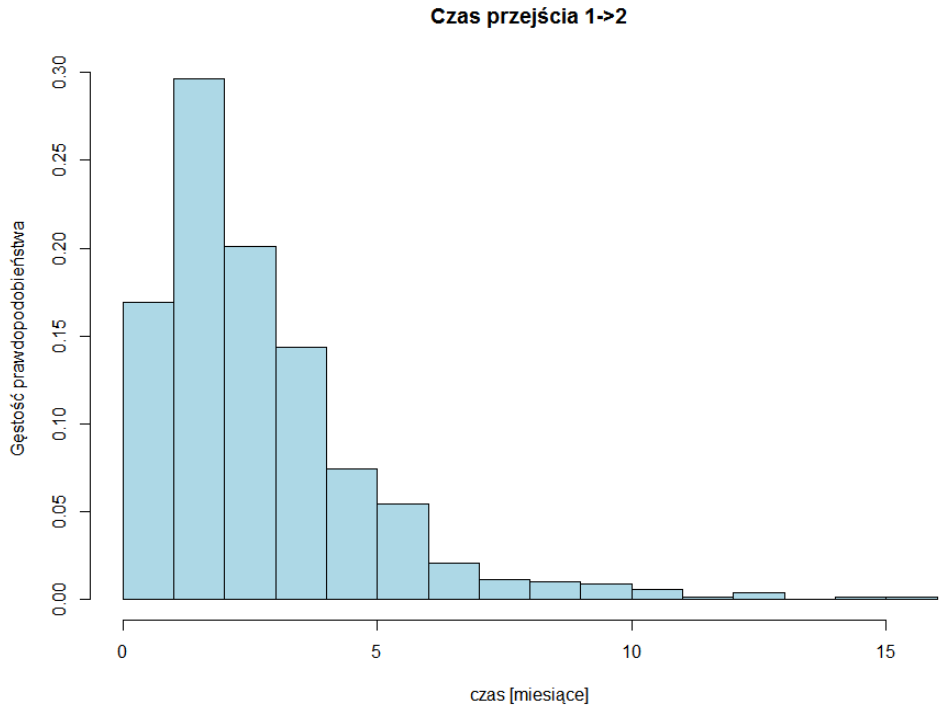
granicznego, a nie poprzez porównanie poziomów bezpieczeństwa przed i po użyciu działań profilaktyczno-naprawczych.

3.6 Przykład obliczeniowy z wykorzystaniem pozyskanej próbki danych

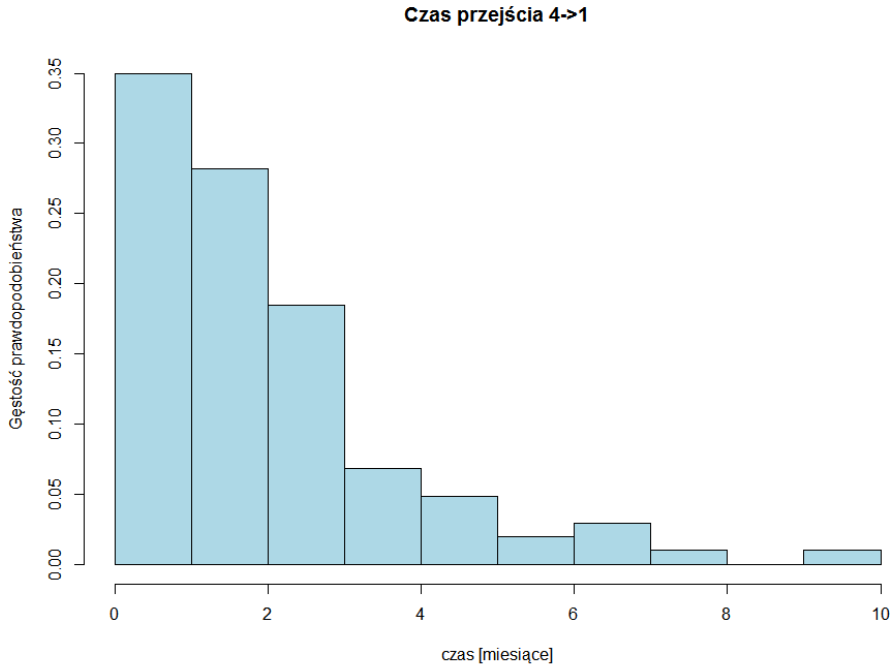
3.6.1 Charakterystyka wykorzystanej próbki danych

Pozyskane próbki danych na potrzeby niniejszej pracy pochodzą od klienta, którego prace wspierają działanie kilku internetowych kantorów wymiany walut. Próbki te wygenerowano na podstawie rejestrowanych w okresach projektowania, testowania oraz utrzymania oprogramowania i zawierają zapis czasów przejść między poszczególnymi stanami w jakich obserwowano badane moduły oprogramowania. Przedstawiono je w formie tabelarycznej, z której można dodatkowo uzyskać informację ile prób było podjętych w przejściu pomiędzy poszczególnymi stanami, dzięki czemu można je wykorzystać do wskazanych obliczeń bazując na zaproponowanych łańcuchach Markowa z rysunków Rys. 29 i Rys. 31. Przykładowy podzbiór wykorzystanej próbki danych do przeprowadzenia obliczeń zamieszczono w Załączniku 2 do niniejszej pracy. Warto zaznaczyć, że próbki te, z uwagi na deklarowaną wrażliwość zawartych informacji pozyskano w formacie częściowo zanonimizowanym i mogą być nieprecyzyjne. Dodatkowo, mając na uwadze, że dane z jednego źródła mogą być niewystarczające, w celu uzyskania statystycznej poprawności, docelowo, w wyniku pracy nad niniejszą pracą powstało narzędzie, do którego możemy wprowadzić alternatywne próbki danych, które zostaną przygotowane we wskazanym formacie lub skorzystać z generatorów, które pozwolą zasymulować potencjalne czasy przejść pomiędzy stanami we wskazanych modelach do dalszej analizy.

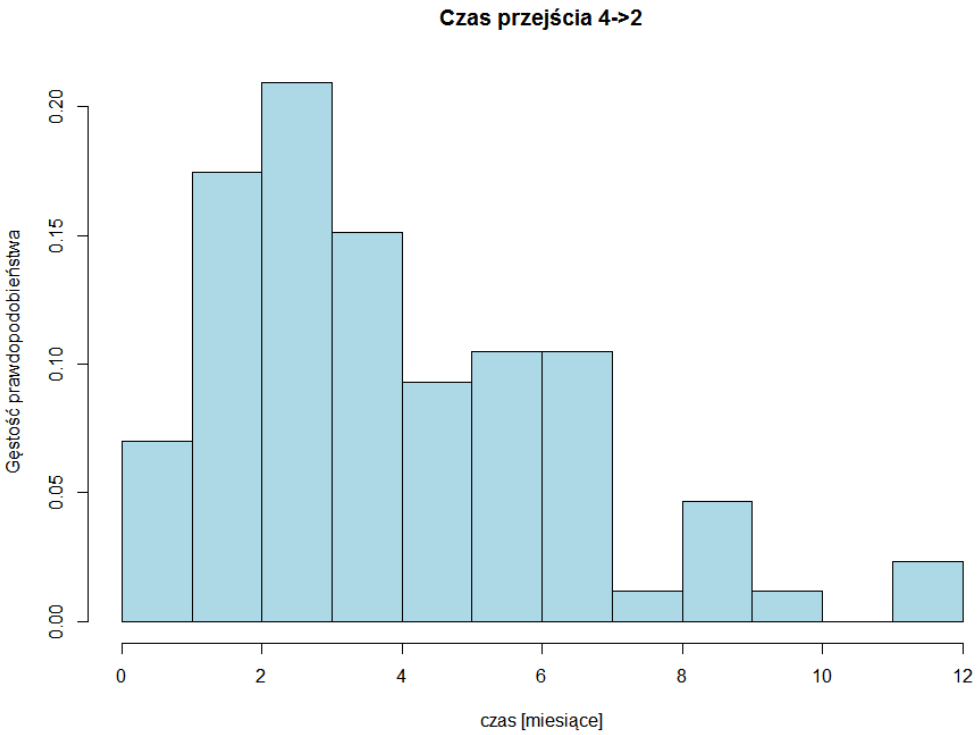
Adaptację modelu rozpoczynamy zatem od przeanalizowania jakie są rozkłady czasów przejść wielokrokowych $1 \rightarrow 2$, $4 \rightarrow 1$, $4 \rightarrow 2$, $4 \rightarrow 3$ z wykorzystanej do analizy próbki danych wejściowych. Histogramy poszczególnych przejść przedstawiono na rysunkach Rys. 35 - Rys. 38.



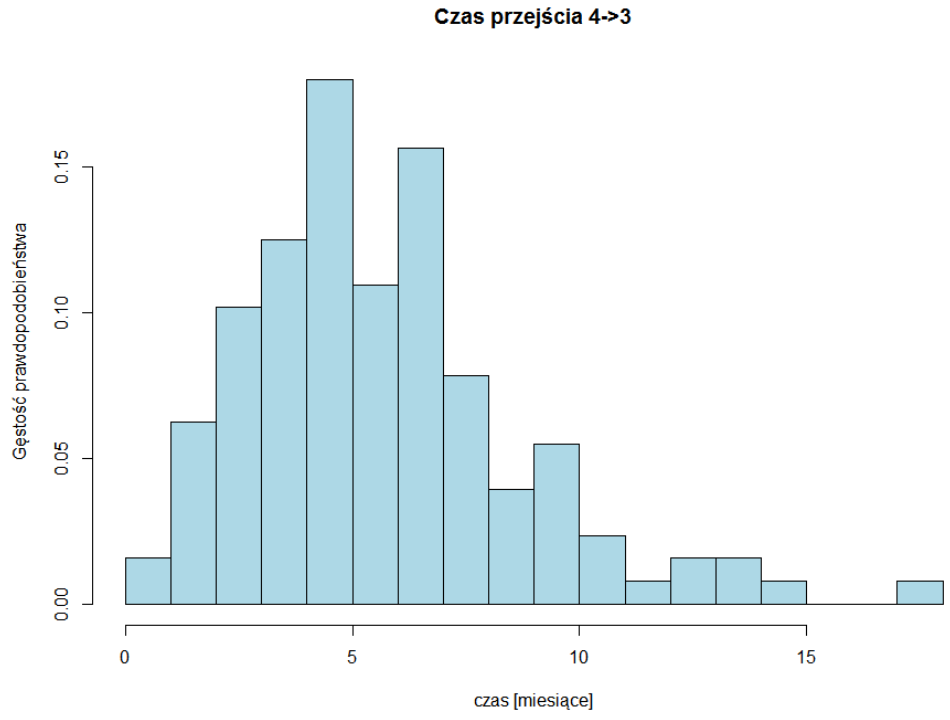
Rys. 35. Histogram czasu przejścia między stanami 1 i 2



Rys. 36. Histogram czasu przejścia między stanami 4 i 1



Rys. 37. Histogram czasu przejścia między stanami 4 i 2



Rys. 38. Histogram czasu przejścia między stanami 4 i 3

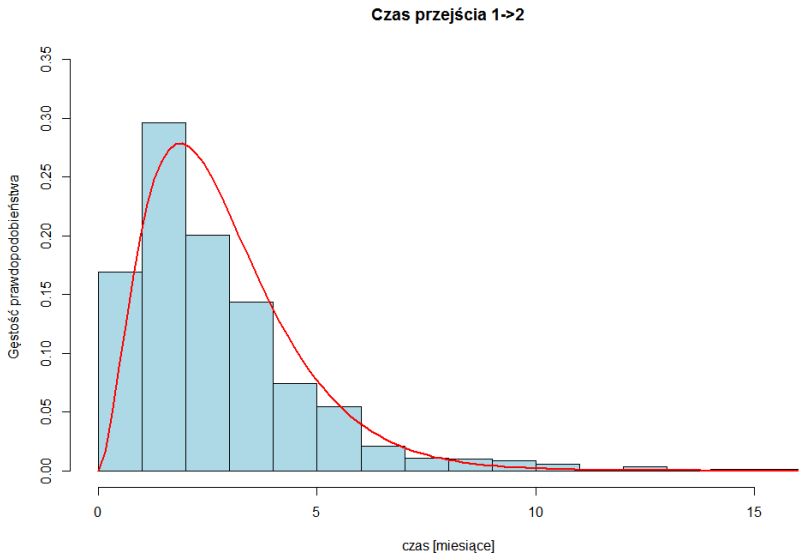
3.6.2 Przykład obliczeń

Histogramy z Rys. 35 - Rys. 38 wskazują, że rozkłady są skośno-symetryczne. Jest to naturalne w zjawiskach, gdzie wraz z czasem prawdopodobieństwo wystąpienia danego zjawiska jest coraz mniejsze. Poszukiwania rozkładów ograniczymy zatem do rozważenia trzech najczęściej występujących, tzn. Rozkładu Poissona, Rozkładu wykładniczego oraz do Rozkładu Weibulla (z ustalonym parametrem skali $k = 3$). Poniższa tabela przedstawia obliczone parametry poszczególnych przejść dla danych rozkładów, wraz z przedziałami ufności, które są obliczane na poziomie ufności 0,95.

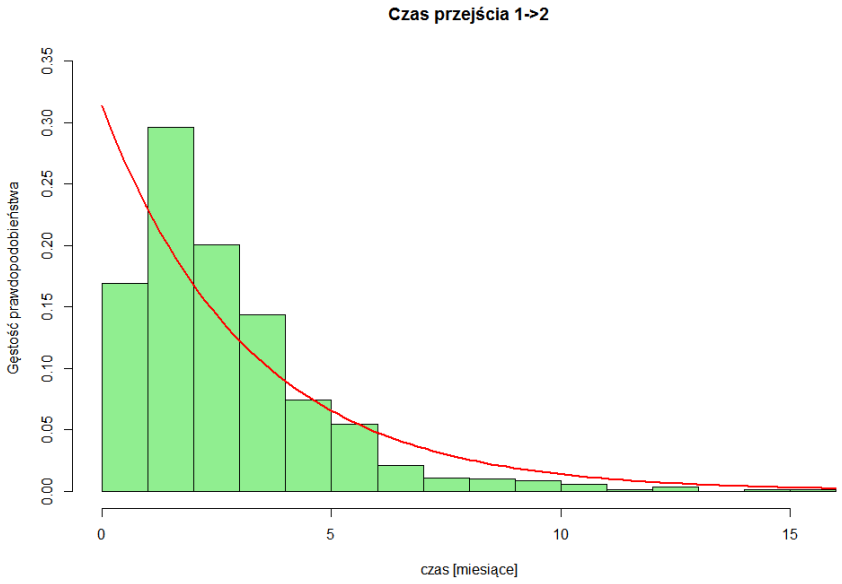
Tabela 7. Parametry poszczególnych rozkładów wraz z przedziałami ufności

	$\mathbf{1 \rightarrow 2}$	$\mathbf{4 \rightarrow 1}$	$\mathbf{4 \rightarrow 2}$	$\mathbf{4 \rightarrow 3}$
Rozkład Poissona	λ =3,188	λ =2,476	λ =4,384	λ =6,148
	[3,073; 3,303]	[2,171; 2,77]	[3,941; 4,826]	[5,719; 6,78]
Rozkład wykładniczy	λ =0,314	λ =0,404	λ =0,228	λ =0,163
	[0,294; 0,334]	[0,33; 0,486]	[0,182; 0,279]	[0,136; 0,192]
Rozkład Weibulla	λ =1,517	λ =1,609	λ =1,255	λ =0,961
	[1,455; 1,58]	[1,389; 1,828]	[1,092; 1,419]	[0,858; 1,064]

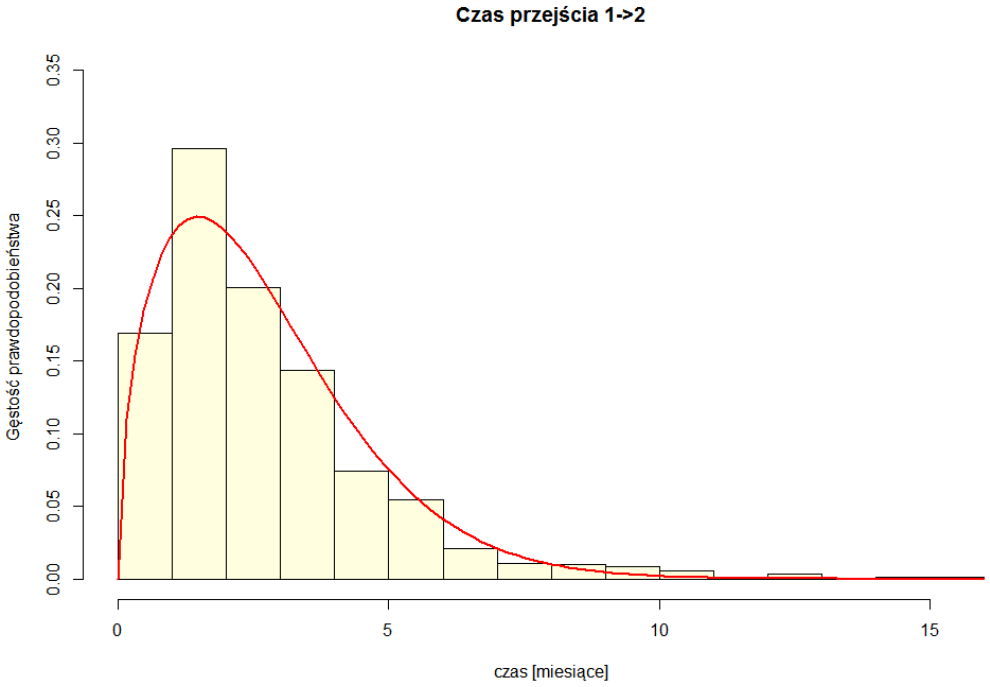
Mając oszacowane parametry rozkładów możemy porównać jak wyglądają wykresy poszczególnych rozkładów w odniesieniu do histogramów wynikających z danych. Poniżej przedstawiamy wykresy.



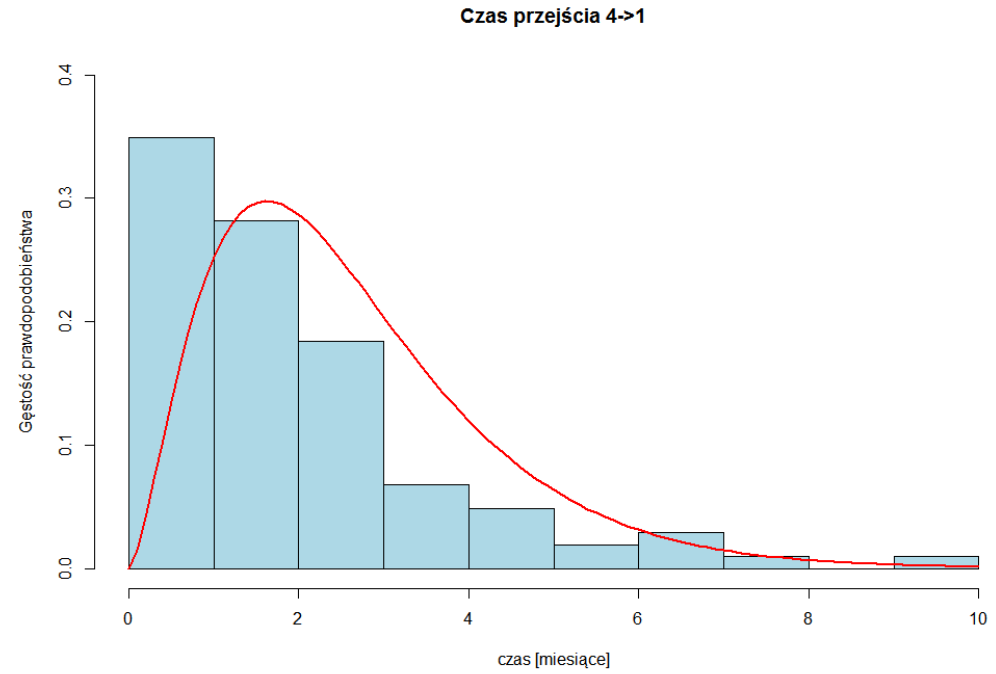
Rys. 39. Histogram czasu przejścia między stanami 1 i 2 wraz z oszacowanym Rozkładem Poissona



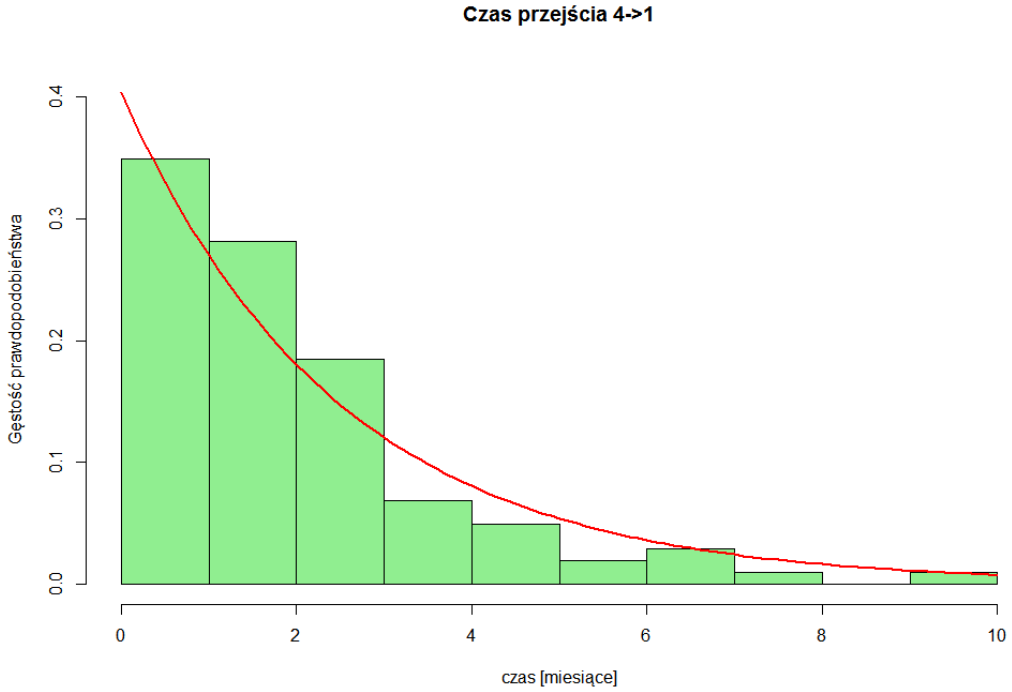
Rys. 40. Histogram czasu przejścia między stanami 1 i 2 wraz z oszacowanym Rozkładem wykładniczym



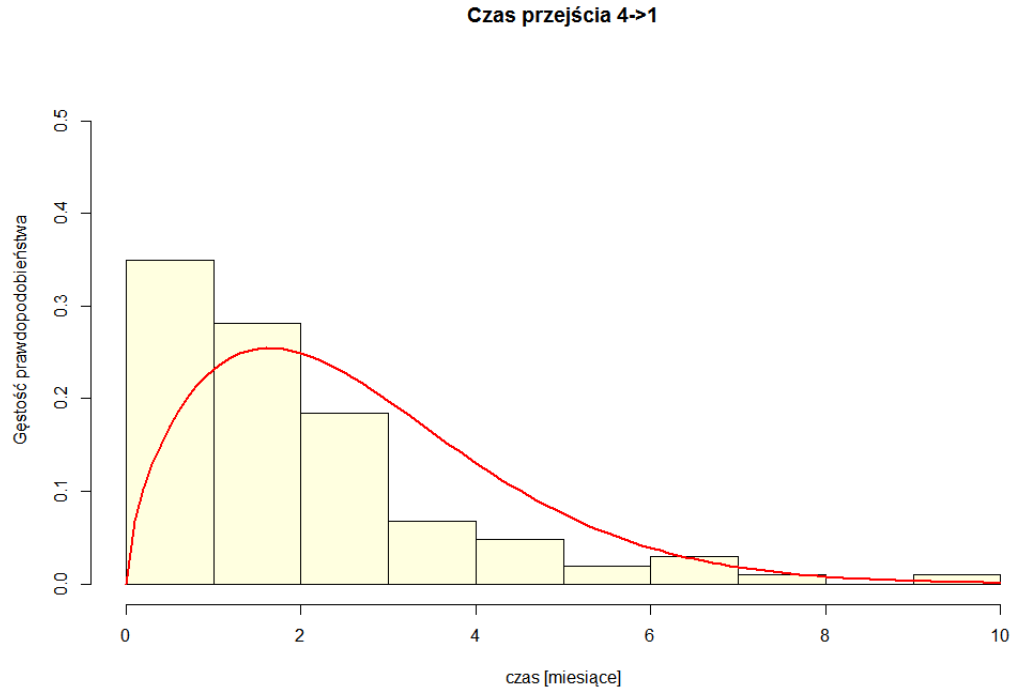
Rys. 41. Histogram czasu przejścia między stanami 1 i 2 wraz z oszacowanym Rozkładem Weibulla



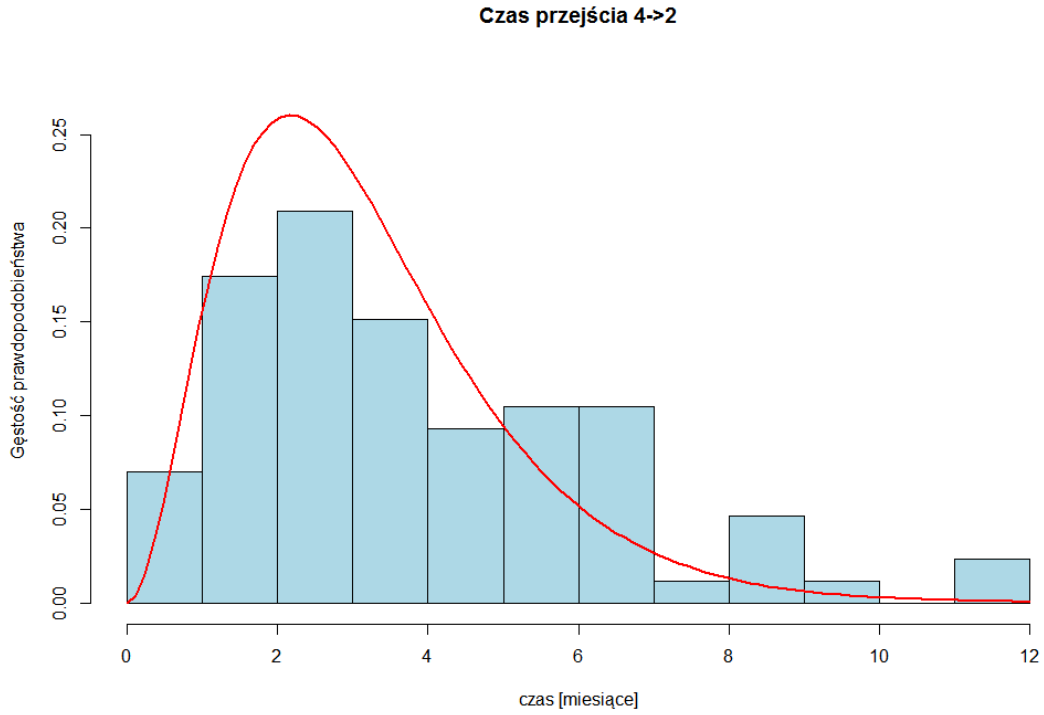
Rys. 42. Histogram czasu przejścia między stanami 4 i 1 wraz z oszacowanym Rozkładem Poissona



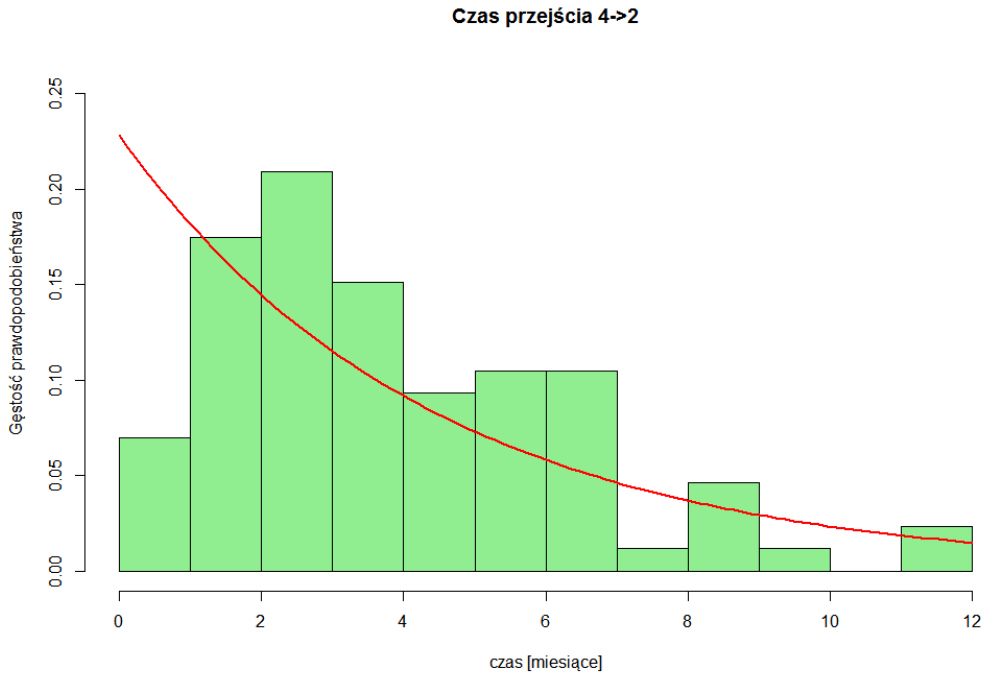
Rys. 43. Histogram czasu przejścia między stanami 4 i 1 wraz z oszacowanym Rozkładem wykładniczym



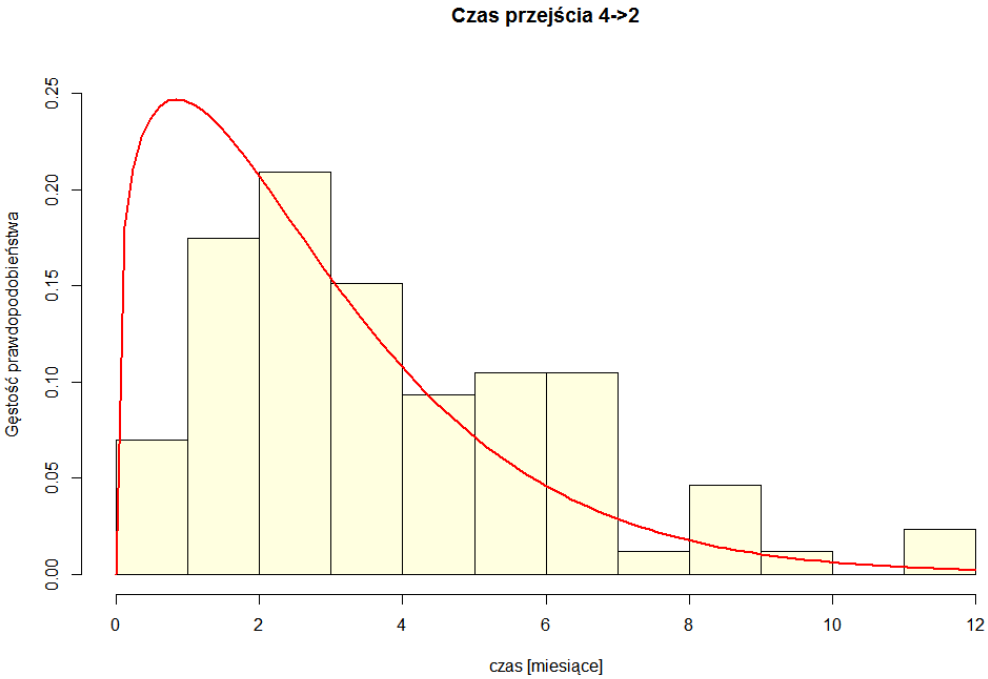
Rys. 44. Histogram czasu przejścia między stanami 4 i 1 wraz z oszacowanym Rozkładem Weibulla



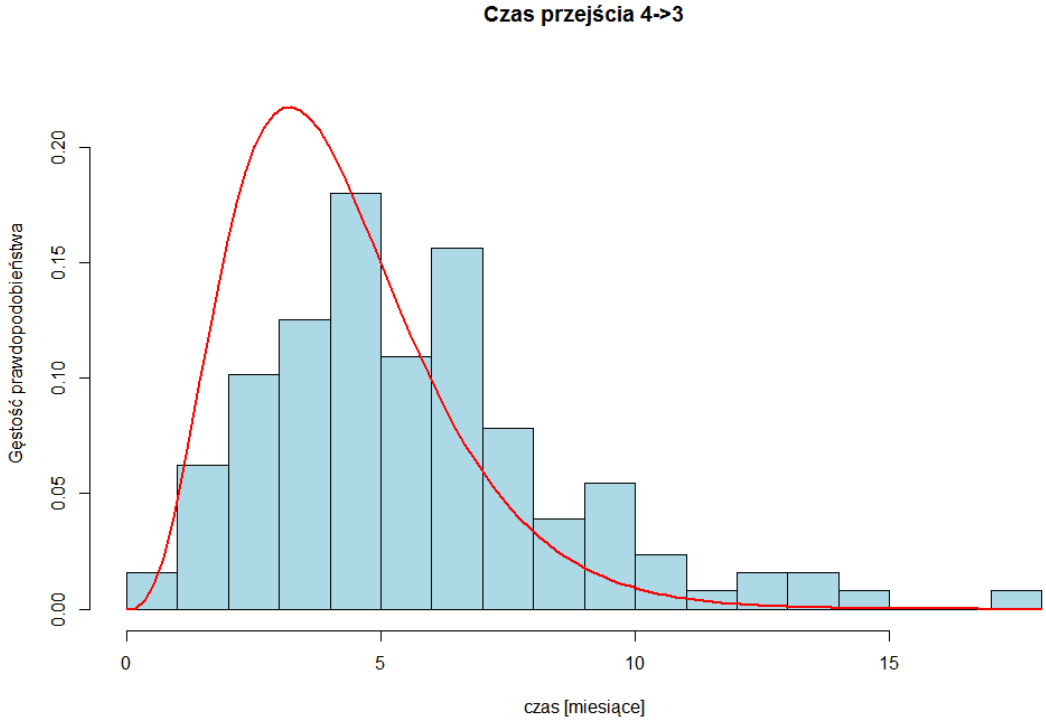
Rys. 45. Histogram czasu przejścia między stanami 4 i 2 wraz z oszacowanym Rozkładem Poissona



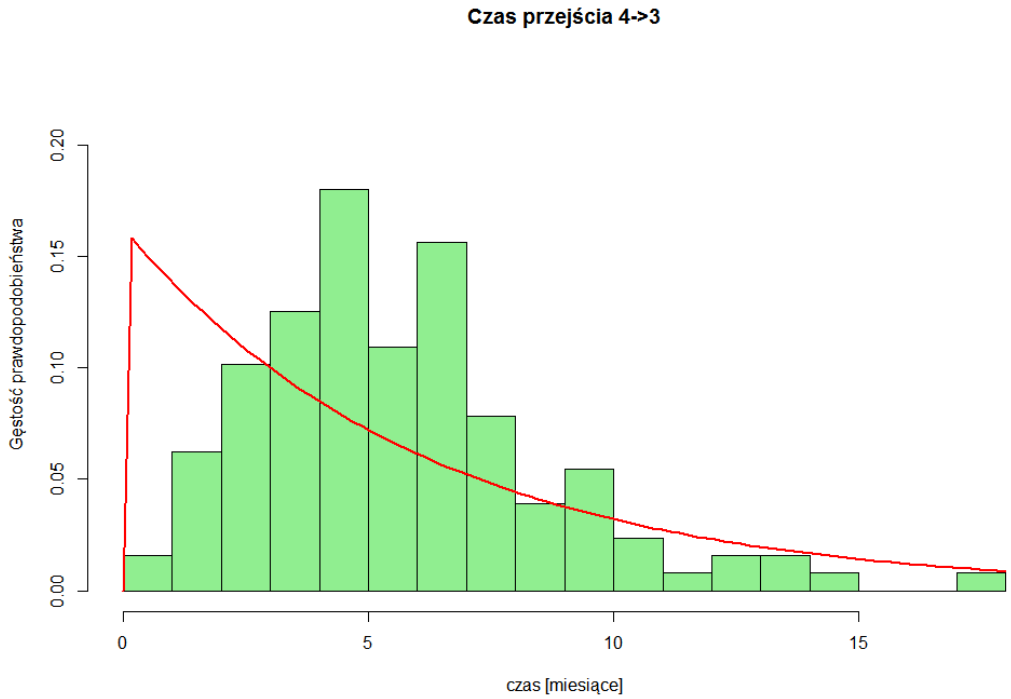
Rys. 46. Histogram czasu przejścia między stanami 4 i 2 wraz z oszacowanym Rozkładem wykładniczym



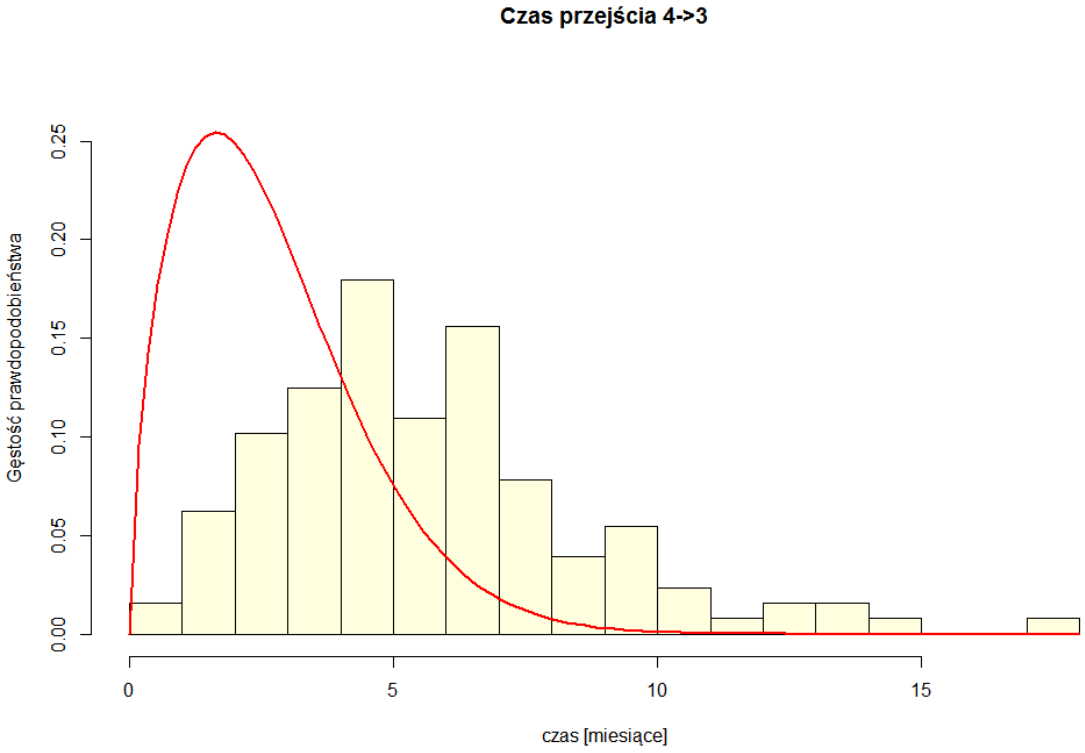
Rys. 47. Histogram czasu przejścia między stanami 4 i 2 wraz z oszacowanym Rozkładem Weibulla



Rys. 48. Histogram czasu przejścia między stanami 4 i 3 wraz z oszacowanym Rozkładem Poissona



Rys. 49. Histogram czasu przejścia między stanami 4 i 3 wraz z oszacowanym Rozkładem wykładniczym



Rys. 50. Histogram czasu przejścia między stanami 4 i 3 wraz z oszacowanym Rozkładem Weibulla

Kolejnym krokiem jest wyznaczenie wartości oczekiwanej danych rozkładów. Wartości te zebrano w Tabela 8.

Tabela 8. Wartości oczekiwane poszczególnych rozkładów

	$\mathbf{1 \rightarrow 2}$	$\mathbf{4 \rightarrow 1}$	$\mathbf{4 \rightarrow 2}$	$\mathbf{4 \rightarrow 3}$
Rozkład Poissona	$\mathbb{E}[X] = 3{,}3796$	$\mathbb{E}[X] = 2{,}1650$	$\mathbb{E}[X] = 4{,}5581$	$\mathbb{E}[X] = 5{,}914$
Rozkład wykładniczy	$\mathbb{E}[X] = 3{,}3796$	$\mathbb{E}[X] = 2{,}1650$	$\mathbb{E}[X] = 4{,}5581$	$\mathbb{E}[X] = 5{,}914$
Rozkład Weibulla	$\mathbb{E}[X] = 2{,}708$	$\mathbb{E}[X] = 2{,}6739$	$\mathbb{E}[X] = 2{,}8156$	$\mathbb{E}[X] = 3{,}0045$

Widzimy, że wartości oczekiwane z Rozkładów Weibulla znacząco się różnią od dwóch pozostałych rozkładów (które są niemal identyczne). Możemy zatem przyjąć, że wartości oczekiwane dwóch pierwszych rozkładów są bliższe rzeczywistym wartościom. Ponadto, w dalszej części pracy będziemy potrzebowali dyskretnych wartości oczekiwanych. Zatem możemy w tym miejscu przyjąć założenie o wartościach oczekiwanych, które umieściliśmy w Tabela 9.

Tabela 9. Wartości oczekiwane zaokrąglone na potrzeby dalszych kroków modelu

$\mathbf{1 \rightarrow 2}$	$\mathbf{4 \rightarrow 1}$	$\mathbf{4 \rightarrow 2}$	$\mathbf{4 \rightarrow 3}$
$\mathbb{E}[X] = 3$	$\mathbb{E}[X] = 2$	$\mathbb{E}[X] = 5$	$\mathbb{E}[X] = 6$

Następnie obliczamy prawdopodobieństwa przejścia na podstawie zebranych danych. Prawdopodobieństwa, te obliczamy jako iloraz liczebności przejść w danych stanach do ogólnej liczebności. Prawdopodobieństwa są liczone odpowiednio dla modelu bez oraz z zastosowaniem działań profilaktyczno-naprawczych. Stąd otrzymujemy następujące wartości dla modelu bez zastosowania działań profilaktyczno-naprawczych i bez przejść wielokrokowych:

$$p_{11} = 0{,}08, \quad p_{12} = 0{,}92, \quad p_{21} = 0{,}134, \quad p_{23} = 0{,}497, \quad p_{24} = 0{,}368,$$

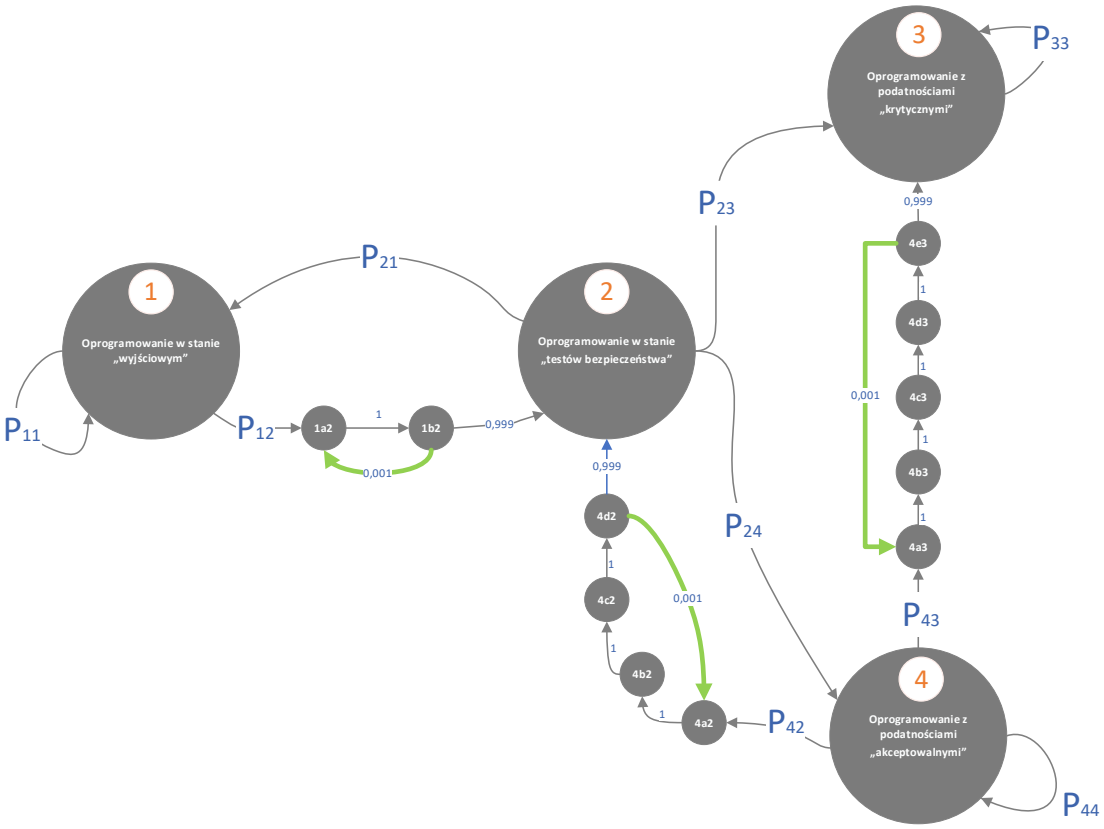
$$p_{33} = 1, \quad p_{42} = 0{,}298, \quad p_{43} = 0{,}443, \quad p_{44} = 0{,}26.$$

Macierz przejścia dla tego modelu z uwzględnieniem przejść wielokrokowych jest następującej postaci:

	$\mathbf{1}$	$\mathbf{1a2}$	$\mathbf{1b2}$	$\mathbf{2}$	$\mathbf{3}$	$\mathbf{4}$	$\mathbf{4a2}$	$\mathbf{4b2}$	$\mathbf{4c2}$	$\mathbf{4d2}$	$\mathbf{4a3}$	$\mathbf{4b3}$	$\mathbf{4c3}$	$\mathbf{4d3}$	$\mathbf{4e3}$
$\mathbf{1}$	0,08	0,92	0	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{1a2}$	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{1b2}$	0	0,001	0	0,999	0	0	0	0	0	0	0	0	0	0	0
$\mathbf{2}$	0,134	0	0	0	0,497	0,368	0	0	0	0	0	0	0	0	0
$\mathbf{3}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
$\mathbf{4}$	0	0	0	0	0	0,26	0,298	0	0	0	0,443	0	0	0	0
$\mathbf{4a2}$	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
$\mathbf{4b2}$	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
$\mathbf{4c2}$	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
$\mathbf{4d2}$	0	0	0	0,999	0	0	0,001	0	0	0	0	0	0	0	0
$\mathbf{4a3}$	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$\mathbf{4b3}$	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
$\mathbf{4c3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
$\mathbf{4d3}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
$\mathbf{4e3}$	0	0	0	0	0,999	0	0	0	0	0	0,001	0	0	0	0

gdzie pozycje skonstruowane z cyfr i liter reprezentują stany przejściowe.

Schemat łańcucha Markowa bez działań profilaktyczno-naprawczych jest przedstawiony następująco:



Rys. 51. Łańcuch Markowa bez działań profilaktyczno-naprawczych z uwzględnieniem przejść wielokrokowych

Możemy teraz obliczyć poziom bezpieczeństwa oprogramowania oraz intensywność wykrywania podatności korzystając z Twierdzenie 3 oraz Twierdzenie 4. Po obliczeniach otrzymujemy następujące wskaźniki:

- Poziom bezpieczeństwa = 0,574,
- Intensywność wykrywania podatności = 8,159 [jednostki czasu].

Przechodząc do modelu z uwzględnieniem zastosowania działań profilaktyczno-naprawczych otrzymujemy następujące prawdopodobieństwa przejścia łańcucha Markowa dla modelu ogólnego:

$$p_{11} = 0{,}08, \quad p_{12} = 0{,}92, \quad p_{21} = 0{,}134, \quad p_{23} = 0{,}497,$$

$$p_{24} = 0{,}368, \quad p_{31} = 0{,}756, \quad p_{33} = 0{,}244, \quad p_{41} = 0{,}263$$

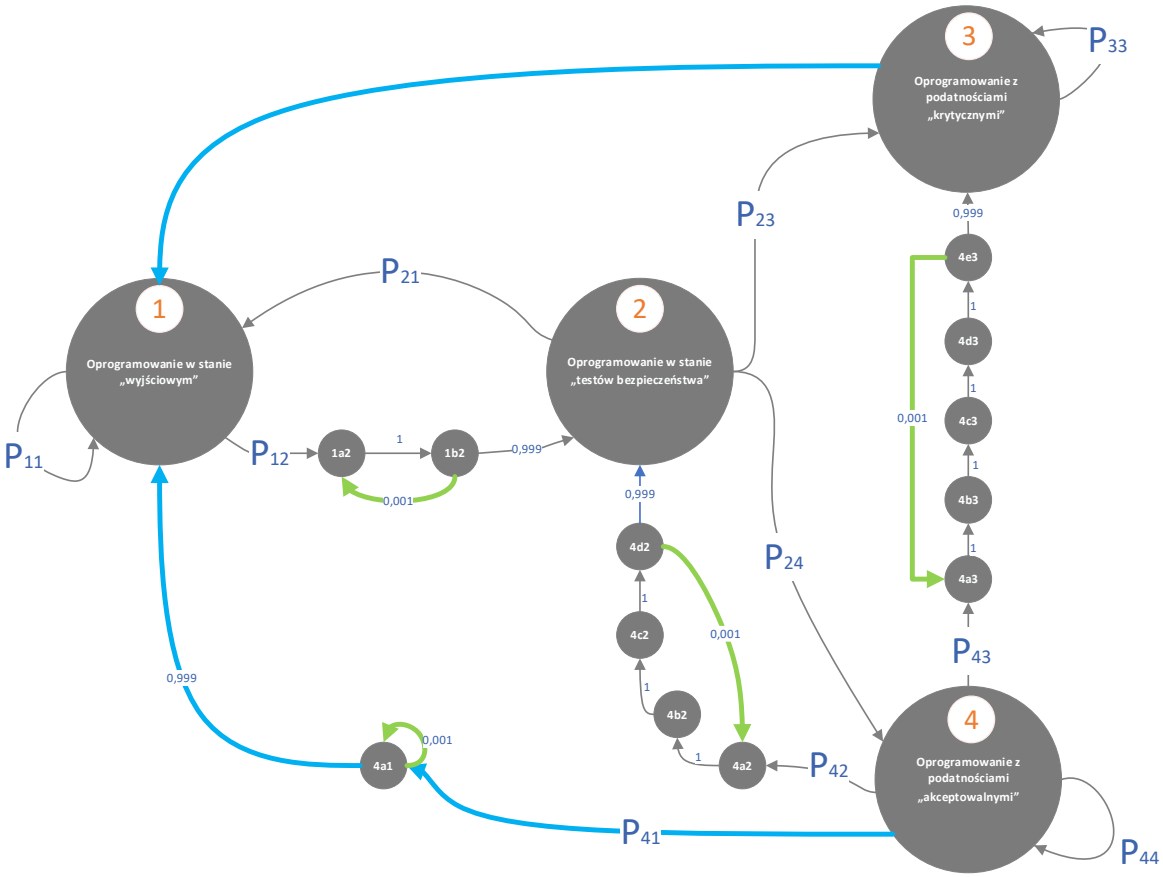
$$p_{42} = 0{,}219, \quad p_{43} = 0{,}327, \quad p_{44} = 0{,}191.$$

Macierz przejścia dla tego modelu z uwzględnieniem przejść wielokrokowych jest następującej postaci:

	1	**1a2**	**1b2**	**2**	**3**	**4**	**4a1**	**4a2**	**4b2**	**4c2**	**4d2**	**4a3**	**4b3**	**4c3**	**4d3**	**4e3**
1	0,08	0,92	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1a2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1b2	0	0,001	0	0,999	0	0	0	0	0	0	0	0	0	0	0	0
2	0,134	0	0	0	0,497	0,368	0	0	0	0	0	0	0	0	0	0
3	0,756	0	0	0	0,244	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0,191	0,263	0,219	0	0	0	0,327	0	0	0	0
4a1	0,999	0	0	0	0	0	0,001	0	0	0	0	0	0	0	0	0
4a2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
4b2	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4c2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
4d2	0	0	0	0,999	0	0	0	0,001	0	0	0	0	0	0	0	0
4a3	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
4b3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
4c3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4d3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
4e3	0	0	0	0	0,999	0	0	0	0	0	0	0,001	0	0	0	0

gdzie pozycje skonstruowane z cyfr i liter reprezentują stany pośrednie.

Natomiast schemat łańcucha Markowa z zastosowaniem działań profilaktyczno-naprawczych jest przedstawiony na Rys. 52.

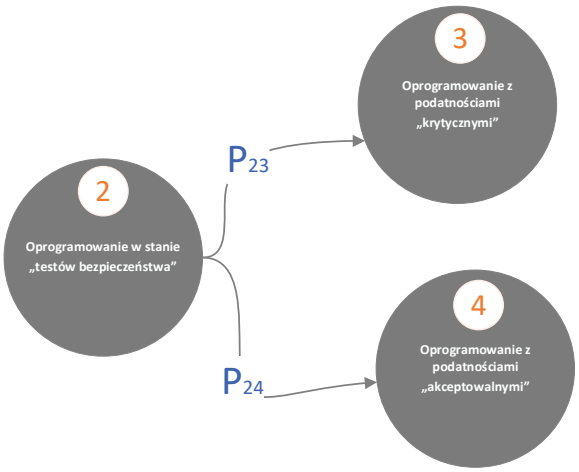


Rys. 52. Łańcuch Markowa z zastosowaniem działań profilaktyczno-naprawczych i z uwzględnieniem przejść wielokrokowych

Możemy teraz obliczyć skuteczność działań profilaktyczno-naprawczych korzystając z Twierdzenie 5. Po obliczeniach otrzymujemy, że rozkład graniczny stanu 1 $\pi_1 = 0{,}256$, natomiast rozkład graniczny stanu 4 wynosi $\pi_4 = 0{,}049$. Zatem można uznać, że zastosowane działania profilaktyczno-naprawcze były skuteczne.

4. Autorskie rozwiązanie problemu klasyfikacji podatności na podstawie danych historycznych z bazy danych podatności

Rozdział czwarty stanowi komplementarną część charakterystyki autorskich modeli przedstawionych w poprzedniej części pracy, koncentrując się na szczegółowym rozważeniu przypadku przyporządkowania nowo-wykrytej podatności do zdefiniowanych wcześniej kategorii podatności, przy wykorzystaniu mechanizmów uczenia maszynowego. Na potrzeby niniejszej pracy, zakładając wykorzystanie wcześniej zdefiniowanych modeli dla rozpatrywanego oprogramowania dedykowanego oraz bazując na empirycznym podejściu do wykonywania testów bezpieczeństwa, można zaobserwować ich częstym wynikiem jest odkrycie podatności, której nie znajdziemy w referencyjnych bazach danych podatności, takich jak np. *NVD*. W takiej bazie danych, każda zarejestrowana podatność jest opisana za pomocą licznych parametrów i metadanych, obejmujących m. in. wartości pojedynczych składowych jej wektora (jak opisano w podrozdziale 2.2.1.1) oraz oszacowany stopień zagrożenia (*Base score*), określony w skali od 0-10, wyliczony na bazie gotowych formuł w systemie scoringowym CVSS. Można go wykorzystać w celu przydzielenia podatności do ustalonych wcześniej kategorii. W niniejszych rozważaniach przyjęto subiektywnie, iż podatności, których wartość stopnia zagrożenia mieści się w przedziale [0; 5] klasyfikuje się jako podatności akceptowalne, natomiast w przedziale (5,10] jako podatności krytyczne.



Rys. 53 Fragment łańcucha Markowa przedstawiający proces klasyfikacji podatności

4.1 Opis rozwiązania

W ujęciu ogólnym, aby wskazać czy wykryta na etapie testów podatność oprogramowania jest krytyczna czy akceptowalna, w pierwszej kolejności należy odwołać się do referencyjnej bazy danych podatności, w celu odczytania wartości ww. wskaźnika.

W przypadku gdy wykryta podatność nie była wcześniej nigdzie zarejestrowana, należy określić wszystkie wymagane wartości składowych jej wektora, aby możliwe było skorzystanie z formuły systemu scoringowego CVSS, w celu oszacowania jej stopnia zagrożenia. Jeśli nie jesteśmy jednak w stanie określić wszystkich wartości składowych wektora podatności, ale możemy wskazać wartości dla kilku wybranych, aby przyporządkować podatność do danej kategorii możliwe jest skorzystanie z możliwości uczenia maszynowego, bazując na próbce danych z prawdziwej bazy danych podatności, którą wykorzystać można do nauczenia modelu (zobrazowanie z przykładowego podzbioru wykorzystanej próbki danych z bazy danych podatności *NVD*, na potrzeby nauczenia modelu, zamieszczono w Załączniku 3 do niniejszej pracy). Następnie mając nauczony model, dla zadanego podzbioru parametrów podatności jesteśmy w stanie go wykorzystać, w celu wskazania czy badana podatność jest krytyczna czy akceptowalna.

W celu dokonania precyzyjnej klasyfikacji podatności zadanie klasyfikacyjne podatności oparliśmy na 4 modelach: regresji logistycznej, analizy dyskryminacyjnej liniowej, analizy dyskryminacyjnej kwadratowej oraz metodzie k najbliższych sąsiadów. Dla pierwszych trzech modeli sposób postępowania jest podobny. Dokładniej, zbiór danych dzielimy na k podzbiorów, przy czym $k-1$ podzbiorów traktujemy jako zbiór uczący/treningowy, a jeden podzbiór jako zbiór testowy. Obliczamy wskaźnik błędu przy użyciu poniższego wzoru

$$\text{error rate} = \frac{\text{liczba błędnych dopasowań w zbiorze testowym}}{\text{liczba elementów w zbiorze testowym}}.$$

Następnie, pierwszych $k-2$ podzbiorów oraz k – ty podzbiór traktujemy jako zestaw uczący treningowy, a zbiór $k-1$ jako zbiór testowy. Wskaźnik błędu obliczamy w sposób analogiczny jak powyżej. W kolejnych krokach podziału dokonujemy zgodnie z poniższą tabelą, gdzie przez U rozumiemy podzbiór uczący/treningowy, natomiast przez T podzbiór testowy.

Tabela 10. Podział k - krotnej walidacji krzyżowej

	Krok 1	**Krok 2**	**Krok 3**	**Krok4**	**…**	**Krok $k-3$**	**Krok $k-2$**	**Krok $k-1$**	**Krok k**
Podzbiór 1	U	U	U	U	…	U	U	U	T

Podzbiór 2	U	U	U	U	…	U	U	T	U
Podzbiór 3	U	U	U	U	…	U	T	U	U
Podzbiór 4	U	U	U	U	…	T	U	U	U
…	…	…	…	…	…	…	…	…	…
Podzbiór $k-3$	U	U	U	T	…	U	U	U	U
Podzbiór $k-2$	U	U	T	U	…	U	U	U	U
Podzbiór $k-1$	U	T	U	U	…	U	U	U	U
Podzbiór k	T	U	U	U	…	U	U	U	U

Z każdego kroku uzyskujemy wskaźnik błędu oraz odpowiednie współczynniki modelu. Ostateczny wskaźnik błędu uzyskujemy poprzez uśrednienie wskaźników błędu z każdego kroku, podobnie wyliczamy ostateczne współczynniki modelu. Jeśli zbiór danych składa się z liczby obserwacji która nie jest podzielna przez k wówczas ostatni podzbiór odpowiednio powiększamy. Opisana metoda nazywa się **k – krotną walidacją krzyżową**. Oczywiście pozostaję pytanie w jaki sposób dobrać k, w literaturze można znaleźć, że optymalny wybór to wartość z przedziału 5-25. W tym celu dla każdego z trzech pierwszych modeli wybierzemy tą wartość k dla której wskaźnik błędu jest najmniejszy, pamiętając przy tym żeby wariancja błędu nie była zbyt duża.

4.2 Wykorzystane metody klasyfikacji

W niniejszym podrozdziale przedstawiono opis metod, które wykorzystano do klasyfikacji podatności, korzystając z następującej pozycji z literatury: [91].

4.2.1 Regresja logistyczna

W dalszej części pracy będziemy korzystali z regresji logistycznej, której wyjście jest binarne, opis tej metody ograniczymy jedynie do tego przypadku.

Będziemy chcieli określić regułę decyzyjną zależną od p zmiennych wejściowych do zmiennej wyjściowej, która przyjmuje dwie wartości. Dokładniej, chcemy określić odwzorowanie

$$\mathbb{R}^p \ni x \to y \in \{-1,1\} \tag{4.1}$$

na podstawie zbioru danych $(x_1, y_1), \dots, (x_N, y_N) \in \mathbb{R}^p \times \{-1,1\}$, które minimalizuje prawdopodobieństwo błędu.

Załóżmy, że dla $x \in \mathbb{R}^p$, prawdopodobieństwo, że $y = 1$ i $y = -1$ jest odpowiednio wyrażone przez

$$\frac{\exp(\beta_0 + x\beta)}{1 + \exp(\beta_0 + x\beta)} \quad oraz \quad \frac{1}{1 + \exp(\beta_0 + x\beta)} \tag{4.2}$$

dla pewnych $\beta_0 \in \mathbb{R}$ oraz $\beta \in \mathbb{R}^p$. Możemy wówczas zapisać prawdopodobieństwo, że $y \in \{-1,1\}$ jako

$$\frac{1}{1 + \exp[-y(\beta_0 + x\beta)]}. \tag{4.3}$$

Naszym zadaniem jest wyznaczenie estymatorów parametrów β_0 oraz β. Do ich wyznaczenia korzystamy z wcześniej opisanej metody największej wiarygodności. Tzn. z obserwacji $(x_1, y_1), \dots, (x_N, y_N) \in \mathbb{R}^p \times \{-1,1\}$, poprzez maksymalizację funkcji wiarygodności

$$L(\beta_0, \beta) = \prod_{i=1}^{N} \frac{1}{1 + \exp[-y_i(\beta_0 + x_i\beta)]} \tag{4.4}$$

lub poprzez minimalizację logarytmu funkcji wiarygodności

$$l(\beta_0, \beta) = \sum_{i=1}^{N} \log(1 + v_i), \tag{4.5}$$

otrzymujemy estymację parametrów β_0 oraz β, gdzie $v_i = \exp[-y_i(\beta_0 + x_i\beta)]$, $i = 1, \dots, N$.

Zauważmy na koniec, że klasyfikacja metoda regresji logistycznej nie jest możliwa w przypadku idealnie rozdzielonych danych.

4.2.2 Analiza dyskryminacyjna

Podobnie jak w przypadku regresji logistycznej, będziemy poszukiwali odwzorowania $x \in \mathbb{R}^p \longmapsto y \in \{-1,1\}$, aby zminimalizować prawdopodobieństwo błędu, biorąc pod uwagę obserwacje $x_1, \dots, x_N \in \mathbb{R}^p$, $y_1, \dots, y_N \in \{-1,1\}$. W tym paragrafie będziemy zakładali, że rozkład zmiennej losowej $X = (X_1, \dots, X_p) \in \mathbb{R}^p$ pod warunkiem, że zmienna losowa $Y \in \{-1,1\}$ ma rozkład normalny o wektorze wartości oczekiwanych $\mu_{\pm 1}$ i macierzy kowariancji $\Sigma_{\pm 1}$, tzn. $(X|Y = \pm 1) \sim N(\mu_{\pm 1}, \Sigma_{\pm 1})$. W takim wypadku gęstością prawdopodobieństwa rozkładu związanego ze zmienną losową $(X|Y = \pm 1)$ jest poniższe funkcja:

$$f_{\pm 1}(x) = \frac{1}{\sqrt{(2\pi)^p \det \Sigma_{\pm 1}}} \exp\left\{-\frac{1}{2}(x - \mu_{\pm 1})^T \Sigma_{\pm 1}^{-1}(x - \mu_{\pm 1})\right\}. \tag{4.6}$$

Dodatkowo wprowadzamy pojęcie prawdopodobieństw a priori zdarzeń: zakładamy, że prawdopodobieństwa odpowiedzi $Y = \pm 1$ są znane przed poznaniem zmiennych x, które nazywamy prawdopodobieństwem a priori. Na przykład, możemy oszacować prawdopodobieństwo odpowiedzi $\pi = \pm 1$ ze stosunku liczby odpowiedzi o etykiecie ± 1 w zbiorze treningowym $y_1, \dots, y_N$ do liczby elementów ogółem. Z drugiej strony odwołujemy się do wielkości

$$\frac{\pi_{\pm 1} f_{\pm 1}(x)}{\pi_1 f_1(x) + \pi_{-1} f_{-1}(x)} \tag{4.7}$$

jako prawdopodobieństwo posteriori zmiennej losowej $Y = \pm 1$ przy danej zmiennej losowej X. Możemy zatem zminimalizować prawdopodobieństwo błędu, szacując $Y = 1$, jeśli

$$\frac{\pi_1 f_1(x)}{\pi_1 f_1(x) + \pi_{-1} f_{-1}(x)} \geq \frac{\pi_{-1} f_{-1}(x)}{\pi_1 f_1(x) + \pi_{-1} f_{-1}(x)}, \tag{4.8}$$

co jest równoważne z

$$\pi_1 f_1(x) \geq \pi_{-1} f_{-1}(x), \tag{4.9}$$

natomiast $Y = -1$ szacujemy w przeciwnym razie (tzn. gdy nierówność jest w przeciwną stronę). Procedura zakłada, że $f_{\pm 1}$ jest zgodne z rozkładem normalnym oraz że znane są: wartość oczekiwana $\mu_{\pm 1}$, macierz kowariancji $\Sigma_{\pm 1}$ oraz znana jest $\pi_{\pm 1}$. W rzeczywistych sytuacjach musimy oszacować te wartości na podstawie danych treningowych. Zatem,

granica decyzyjna oddzielającą obserwacje dla których $Y = 1$ od obserwacji dla których $Y = -1$ jest opisana równaniem

$$\pi_1 f_1(x) = \pi_{-1} f_{-1}(x) \tag{4.10}$$

Podstawiając gęstości prawdopodobieństwa do powyższego równania, a następnie logarytmując stronami otrzymujemy:

$$-(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + (x-\mu_{-1})^T\Sigma_{-1}^{-1}(x-\mu_{-1}) = \log\frac{\det\Sigma_1}{\det\Sigma_{-1}} - 2\log\frac{\pi_1}{\pi_{-1}}. \tag{4.11}$$

Ogólnie rzecz biorąc granica decyzyjna jest hiperpłaszczyzną stopnia drugiego. Mówimy wówczas o **kwadratowej dyskryminacji**. W szczególności, jeśli $\Sigma := \Sigma_1 = \Sigma_{-1}$, to powyższa granica przyjmuje następującą postać:

$$2(\mu_1 - \mu_{-1})^T\Sigma^{-1}x - (\mu_1^T\Sigma^{-1}\mu_1 - \mu_{-1}\Sigma^{-1}\mu_{-1}) = -2\log\frac{\pi_1}{\pi_{-1}}, \tag{4.12}$$

w takim przypadku mówimy granica decyzyjna jest hiperpłaszczyzną i nazywamy ją **dyskryminacja liniową.**

Uwaga. W celu obliczenia parametrów $\mu_{\pm1}$ oraz $\Sigma_{\pm1}$ korzystamy z poniższych wzorów:

$$\mu_{\pm1} = \frac{1}{n_{\pm1}}\sum_{i=1}^{n_1} x_i^{\pm1} \in \mathbb{R}^p, \tag{4.13}$$

$$\Sigma_{\pm1} = \frac{1}{n_{\pm1}-1}\sum_{i=1}^{n_{\pm1}}\left(x_i^{\pm1} - \mu_{\pm1}\right)\left(x_i^{\pm1} - \mu_{\pm1}\right)^T \in \mathbb{R}^{p\times p}, \tag{4.14}$$

gdzie $n_{\pm1}$ jest liczbą obserwacji z etykietą $Y = \pm1$, $x_i^{\pm1} \in \mathbb{R}^p$ jest i-tą obserwacją z etykietą $Y = \pm1$.

4.2.3 Algorytm k - najbliższych sąsiadów (KNN)

W metodzie k-najbliższych sąsiadów, zamiast konstruowania określonej reguły z danych treningowych $(x_1, y_1), \ldots, (x_N, y_N) \in \mathbb{R}^p \times$ (zbiór skończony). Załóżmy, że dana jest nowa obserwacja $x_* \in \mathbb{R}^p$ oraz, że $x_i, i \in S$, są k danymi treningowymi takimi, że odległości między x_i ix_* są najmniejsze, gdzie S jest podzbiorem $\{1, \ldots, n\}$ o mocy k, tzn. $k = \#\{1, \ldots, n\}$. Metoda k - najbliższych sąsiadów przewiduje odpowiedź y_* na podstawie x_* poprzez sprawdzenie jaką jest większość wartości y_i , $i \in S$.

Na przykład, załóżmy, że $N = 5$, $p = 1$ oraz, że dane treningowe są takie jak w Tabela 11.

Tabela 11. Przykładowe dane treningowe dla metody k-najbliższych sąsiadów

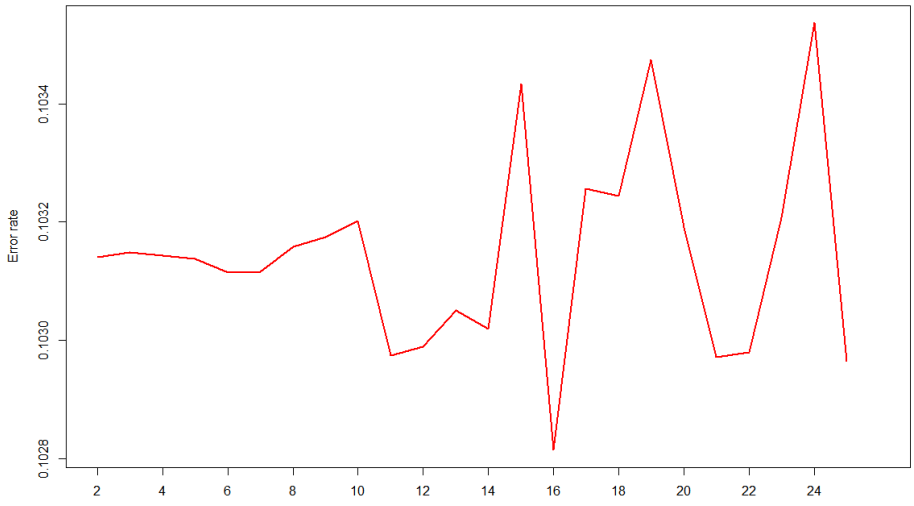
x_i	-2,1	-3,7	1,3	0,4	1,5
y_i	-1	1	0	0	1

Jeśli $k = 3$ oraz $x_* = 1{,}6$, wówczas $S = \{3,4,5\}$ i większość y_i, $i \in S$ stanowi 0, dlatego też przyjmujemy $y_* = 0$. Jeśli $k = 2$ oraz $x_* = -2{,}2$, wówczas $S = \{1,2\}$. Jednak, w tym przypadku większość y_i, $i \in S$ nie jest jednoznaczna. W takim przypadku, usuwamy jeden element ze zbioru S, dla którego odległość między x_i a x_* jest największa. Ponieważ $x_1 = -2{,}1$ jest bliżej położony od punktu $x_* = -2{,}2$ niż punkt x_2, to przyjmy, że $S = \{1\}$ oraz $y_* = -1$.

4.3 Przykład obliczeniowy z wykorzystaniem pozyskanej próbki danych z bazy danych podatności NVD

4.3.1 Regresja logistyczna

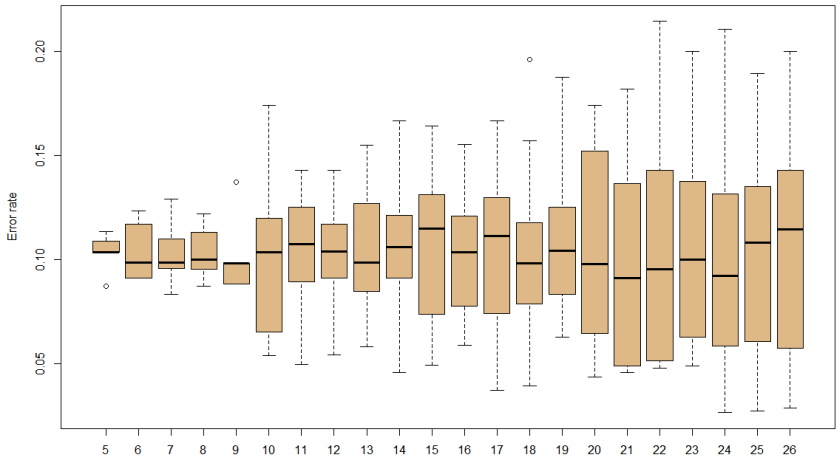
Dane przetestowano i zweryfikowano metodą walidacji krzyżowej dla $k \in \{2, \dots, 25\}$. Na poniższym wykresie przedstawiono wskaźniki błędów dla poszczególnych k.



Rys. 54. Wskaźniki błędów regresji logistycznej dla różnych walidacji krzyżowych

Widzimy, że dla małych k błąd zachowuje się dość stabilnie natomiast wraz ze wzrostem k błąd wykazuje duże wahania. Zatem, aby wskazać optymalną walidację krzyżową należy sprawdzić jeszcze jakie są miary położenia poszczególnych walidacji krzyżowych. Na

poniższym wykresie przedstawiono wykresy pudełkowe dla kolejnych k. Widzimy, że dla $k \leq 9$ błędy poszczególnych kroków w danej walidacji krzyżowej są położonej blisko siebie. Biorąc pod uwagę te dwa wykresy przyjmujemy, że najlepszą walidacją krzyżową (tzn. taką która minimalizuje wskaźnik błędu i wariancje) jest dla $k = 7$.



Rys. 55. Wykresy pudełkowe wskaźników błędów regresji logistycznej dla poszczególnych walidacji krzyżowych

Ostatecznie otrzymujemy, że wskaźnik błędu dla regresji logistycznej wynosi:

$$\text{error rate} \approx 0{,}103.$$

Dodatkowo, parametry tego modelu przedstawiono poniżej:

$$\beta_0 \approx -26{,}319, \quad \beta_1 \approx 18{,}524, \quad \beta_2 \approx 0{,}169, \quad \beta_3 \approx 6{,}777, \quad \beta_4 = 16{,}801.$$

Wstawiając powyższe wartości do wzoru regresji logistycznej otrzymujemy, że prawdopodobieństwo sklasyfikowania podatności jako niebezpiecznej można obliczyć z poniższego wzoru:

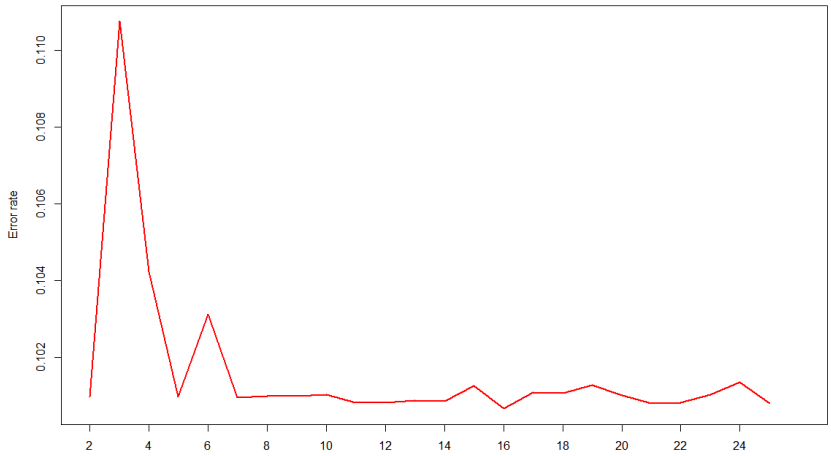
$$\frac{\exp(-26{,}319 + 18{,}524x_1 + 0{,}169x_2 + 6{,}777x_3 + 16{,}801x_4)}{1 + \exp(-26{,}319 + 18{,}524x_1 + 0{,}169x_2 + 6{,}777x_3 + 16{,}801x_4} \tag{4.15}$$

natomiast prawdopodobieństwo, że podatność jest bezpieczna można obliczyć z następującego wzoru:

$$\frac{1}{1 + \exp(-26{,}319 + 18{,}524x_1 + 0{,}169x_2 + 6{,}777x_3 + 16{,}801x_4} \tag{4.16}$$

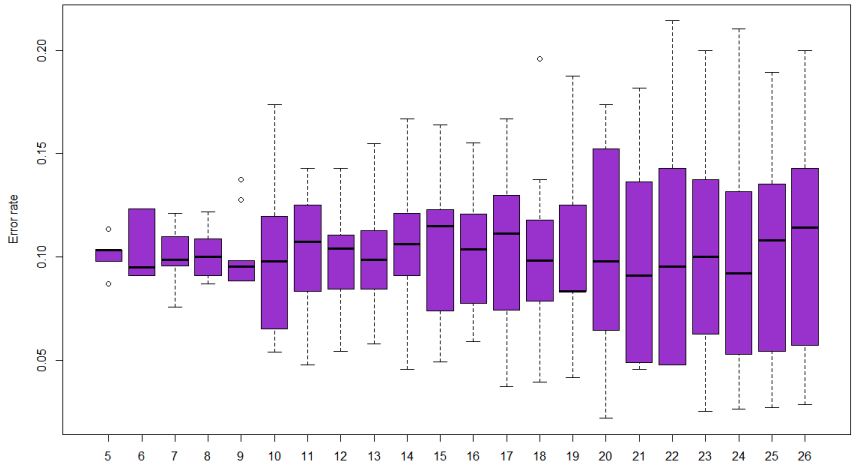
4.3.2 Analiza dyskryminacyjna liniowa

Dane przetestowano i zweryfikowano metodą walidacji krzyżowej dla $k \in \{2, \dots, 25\}$. Na poniższym wykresie przedstawiono wskaźniki błędów dla poszczególnych k.



Rys. 56. Wskaźniki błędów analizy dyskryminacyjnej liniowej dla różnych walidacji krzyżowych

Widzimy, że dla małych k błąd zachowuje się dość niestabilnie natomiast wraz ze wzrostem k błąd wykazuje stabilizację. Zatem, aby wskazać optymalną walidację krzyżową należy sprawdzić jeszcze jakie są miary położenia poszczególnych walidacji krzyżowych. Na poniższym wykresie przedstawiono wykresy pudełkowe dla kolejnych k. Widzimy, że dla $k \leq 9$ błędy poszczególnych kroków w danej walidacji krzyżowej są położonej blisko siebie. Biorąc pod uwagę te dwa wykresy przyjmujemy, że najlepszą walidacją krzyżową (tzn. taką która minimalizuje wskaźnik błędu i wariancje) jest dla $k = 8$.



Rys. 57. Wykresy pudełkowe wskaźników błędów analizy dyskryminacyjnej liniowej dla poszczególnych walidacji krzyżowych

Ostatecznie otrzymujemy, że wskaźnik błędu dla regresji logistycznej wynosi:

$$\text{error rate} \approx 0{,}101.$$

Dodatkowo, parametry tego modelu przedstawiono poniżej:

$$\mu_1 \approx (0{,}988; 0{,}657; 0{,}695; 0{,}365), \quad \mu_{-1} \approx (0{,}967; 0{,}646; 0{,}681; 0{,}074),$$

$$\pi_1 \approx 0{,}574, \qquad \pi_{-1} \approx 0{,}426,$$

$$\Sigma \approx \begin{bmatrix} 0{,}00786 & -0{,}000385 & -0{,}000318 & -0{,}00423 \\ -0{,}000385 & 0{,}00374 & 0{,}0000262 & 0{,}000248 \\ -0{,}000318 & 0{,}0000262 & 0{,}00199 & 0{,}000298 \\ -0{,}00423 & 0{,}000248 & 0{,}000298 & 0{,}0246 \end{bmatrix},$$

$$\Sigma^{-1} \approx \begin{bmatrix} 142{,}04 & 12{,}859 & 18{,}789 & 24{,}0234 \\ 12{,}859 & 269{,}629 & -1{,}464 & -0{,}489 \\ 18{,}789 & -1{,}464 & 505{,}901 & -2{,}897 \\ 24{,}0234 & -0{,}489 & -2{,}897 & 44{,}882 \end{bmatrix}.$$

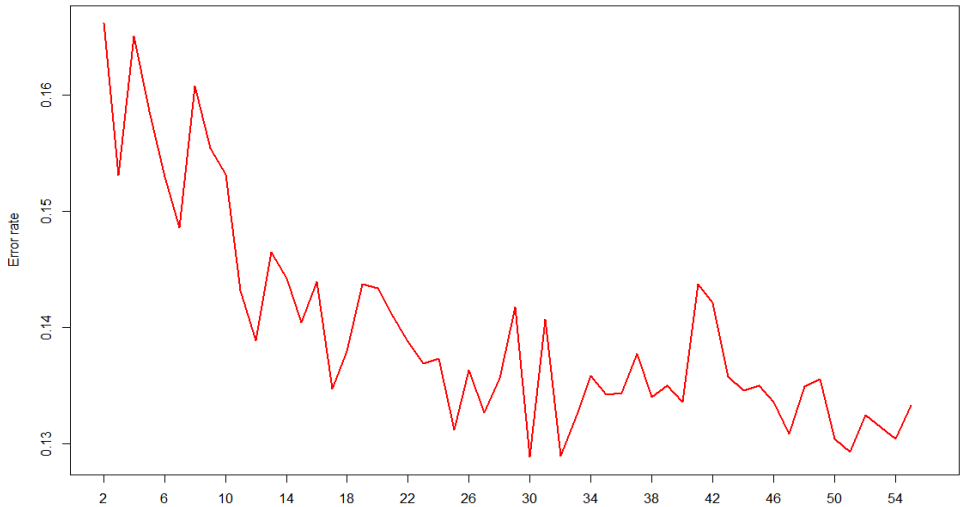
Przy czym Σ była obliczona jako średnia arytmetyczna macierzy kowariancji danych z etykietą 1 oraz macierzy kowariancji danych z etykietą -1. Podstawiając powyższe wartości do wzoru analizy dyskryminacyjnej liniowej otrzymujemy równanie granicy decyzyjnej:

$$20{,}803x_1 + 6{,}304x_2 + 13{,}015x_3 + 27{,}077x_4 - 38{,}739 = 0. \qquad (4.17)$$

Jeśli powyższe równanie przyjmuje wartości dodatnie wówczas zakładamy, że podatność jest krytyczna, w przeciwnym przypadku, że podatność jest akceptowalna.

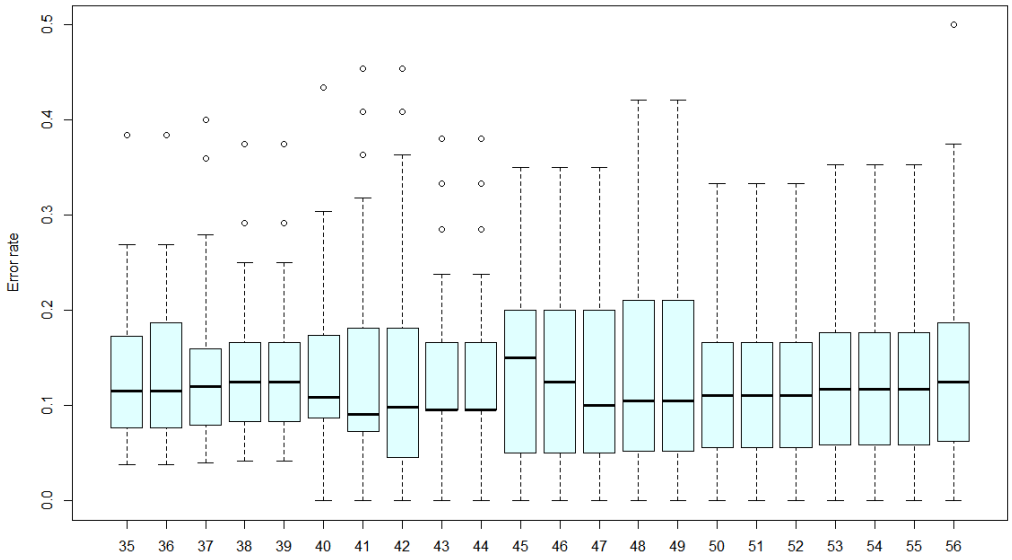
4.3.3 Analiza dyskryminacyjna kwadratowa

Dane przetestowano i zweryfikowano metodą walidacji krzyżowej dla $k \in \{2, \dots, 55\}$. Na poniższym wykresie przedstawiono wskaźniki błędów dla poszczególnych k.



Rys. 58. Wskaźniki błędów analizy dyskryminacyjnej kwadratowej dla różnych walidacji krzyżowych

Widzimy, że dla małych k błąd wykazuje dość duże wahani, natomiast dla $k > 30$ pojawia się pewna stabilizacja. Zatem, aby wskazać optymalną walidację krzyżowa należy sprawdzić jeszcze jakie są miary położenia poszczególnych walidacji krzyżowych, w tym wypadku zrobimy to dla $k \geq 35$ (ze względu na stabilizacje błędu w tym zakresie). Na poniższym wykresie przedstawiono wykresy pudełkowe dla kolejnych k. Widzimy, że dla $k \leq 40$ błędy poszczególnych kroków w danej walidacji krzyżowej są położonej blisko siebie. Biorąc pod uwagę te dwa wykresy przyjmujemy, że najlepszą walidacją krzyżową (tzn. taką która minimalizuje wskaźnik błędu i wariancje) jest dla $k = 38$.



Rys. 59. Wykresy pudełkowe wskaźników błędów analizy dyskryminacyjnej kwadratowej dla poszczególnych walidacji krzyżowych

Ostatecznie otrzymujemy, że wskaźnik błędu dla regresji logistycznej wynosi

$$\text{error rate} \approx 0{,}134.$$

Dodatkowo, parametry tego modelu przedstawiono poniżej:

$$\mu_1 \approx (0{,}988; 0{,}656; 0{,}695; 0{,}365), \quad \mu_{-1} \approx (0{,}967; 0{,}646; 0{,}681; 0{,}074),$$

$$\pi_1 \approx 0{,}579, \qquad \pi_{-1} \approx 0{,}421,$$

$$\Sigma_1 \approx \begin{bmatrix} 0{,}00435 & -0{,}000122 & -0{,}000107 & -0{,}0035 \\ -0{,}000122 & 0{,}00374 & 0{,}000288 & 0{,}00121 \\ -0{,}000107 & 0{,}000288 & 0{,}00122 & 0{,}000435 \\ -0{,}0035 & 0{,}00121 & 0{,}000435 & 0{,}0311 \end{bmatrix},$$

$$\Sigma_{-1} \approx \begin{bmatrix} 0{,}0114 & -0{,}000648 & -0{,}000528 & -0{,}00497 \\ -0{,}000648 & 0{,}00374 & 0{,}000341 & 0{,}00171 \\ -0{,}000528 & 0{,}000341 & 0{,}00277 & 0{,}000163 \\ -0{,}00497 & 0{,}00171 & 0{,}000163 & 0{,}0181 \end{bmatrix},$$

$$\Sigma_1^{-1} \approx \begin{bmatrix} 254{,}677 & 19{,}013 & 16{,}374 & 29{,}182 \\ 19{,}013 & 277{,}193 & 62{,}943 & 12{,}041 \\ 16{,}374 & 62{,}943 & 842{,}471 & -7{,}501 \\ 29{,}182 & 12{,}041 & -7{,}501 & 36{,}048 \end{bmatrix},$$

$$\Sigma_{-1}^{-1} \approx \begin{bmatrix} 100{,}989 & 3{,}509 & 17{,}22 & 27{,}197 \\ 3{,}509 & 283{,}272 & -32{,}679 & -25{,}441 \\ 17{,}22 & -32{,}679 & 368{,}459 & 4{,}495 \\ 27{,}197 & -25{,}441 & 4{,}495 & 65{,}052 \end{bmatrix}.$$

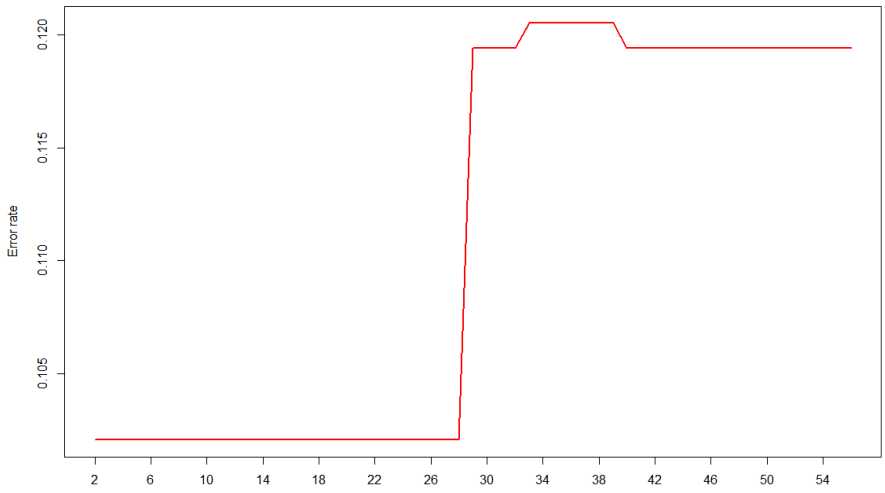
Podstawiając powyższe wartości do wzoru analizy dyskryminacyjnej kwadratowej otrzymujemy równanie granicy decyzyjnej:

$$\begin{gathered} -153{,}688x_1^2 + 6{,}078x_2^2 - 474{,}012x_3^2 + 29{,}003x_4^2 - 31{,}009x_1x_2 + \\ +1{,}693x_1x_3 - 3{,}971x_1x_4 - 191{,}242x_2x_3 - 74{,}964x_2x_4 + 23{,}993x_3x_4 + \\ +345{,}026x_1 + 173{,}751x_2 + 786{,}842x_3 + 53{,}916x_4 - 521{,}815 = 0 \end{gathered} \tag{4.18}$$

Jeśli powyższe równanie przyjmuje wartości dodatnie wówczas zakładamy, że podatność jest krytyczna, w przeciwnym przypadku, ze podatność jest akceptowalna.

4.3.4 Algorytm k - najbliższych sąsiadów (KNN)

W przypadku metody *k* najbliższych sąsiadów sposób postepowania jest trochę inny niż w poprzednich trzech modelach. Ponieważ model KNN nie posiada żadnych własnych parametrów, porównywanie różnych walidacji krzyżowych traci sens w tym przypadku. Zauważmy, że przy prognozie będziemy używali całego zbioru jako dane uczące/treningowe. Zatem aby wyznaczyć wskaźnik błędu skorzystamy ze szczególnej postaci walidacji krzyżowej, w której w każdym kroku jako zbiór testowy przyjmujemy dokładnie jedną obserwację natomiast pozostałe obserwacje służą za zbiór uczący/treningowy. Taka szczególna forma walidacji krzyżowej nazywa się LOOCV (*leave-one-out cross validation*). Przy takiej postaci walidacji krzyżowej nie ma oczywiście sensu rozważać miar położenia wskaźników błędu kroków, ponieważ krokowe błędy będą przyjmowały wartości binarne. Zatem przy doborze optymalnej liczby sąsiadów będziemy polegali jedynie na wskaźnikach błędu poszczególnej liczby sąsiadów w metodzie KNN. Poniżej przedstawiliśmy wykres wskaźników błędu dla poszczególnych liczb sąsiadów.



Rys. 60. Wykres błędów metody KNN dla różnej liczby sąsiadów

Analizując Rys. 60, widzimy, że wskaźnik błędu jest taki sam dla $k < 27$ zatem możemy przyjąć, że $k = 5$. Wówczas otrzymujemy, że wskaźnik błędu wynosi

$$\text{error rate} \approx 0{,}102.$$

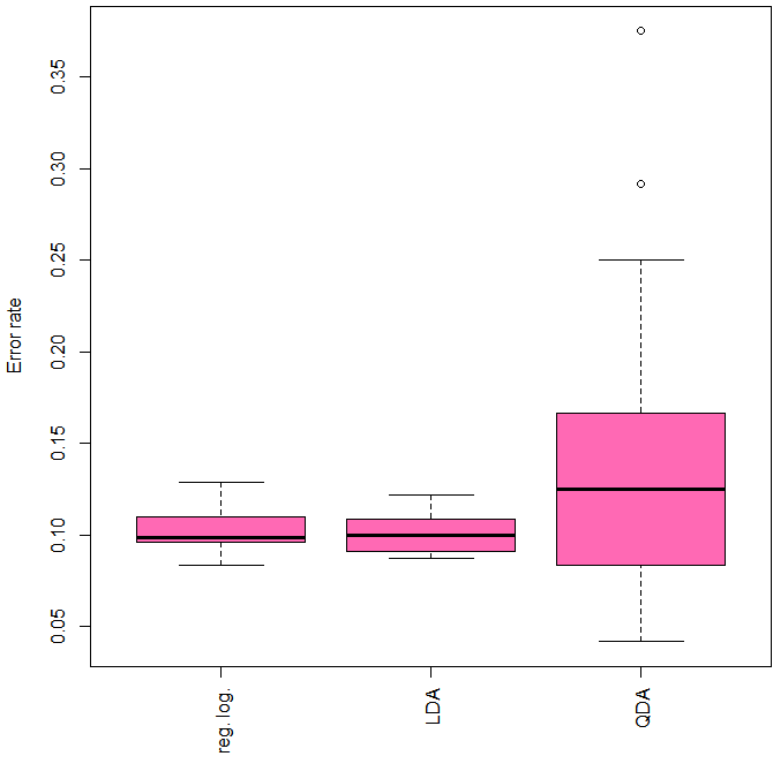
4.4 Selekcja modelu

Na zakończenie pozostaje nam wskazanie który model jest optymalny. Zacznijmy od zestawienia wskaźników błędu.

Tabela 12. Zestawienia wskaźników błędów poszczególnych modeli

Model	Error rate
Regresja logistyczna	0,103
Analiza dyskryminacyjna liniowa	0,101
Analiza dyskryminacyjna kwadratowa	0,134
KNN	0,102

Analizując Tabela 12 możemy dojść do wniosku, że optymalna będzie analiza dyskryminacyjna liniowa w grupie modeli parametrycznych oraz model KNN. Jednak, aby mieć pełen obraz powinniśmy przeanalizować miary położenia pierwszych trzech modeli (modelu KNN nie możemy rozważać w kategorii miary położenia). Miary położenia przedstawiono na poniższym wykresie pudełkowym. Z Rys. 61 możemy wywnioskować, że najmniejszy rozrzut posiada model analizy dyskryminacyjnej liniowej. Ostatecznie możemy przyjąć, że modele LDA oraz KNN są najbardziej adekwatne do rozważanej problematyki.

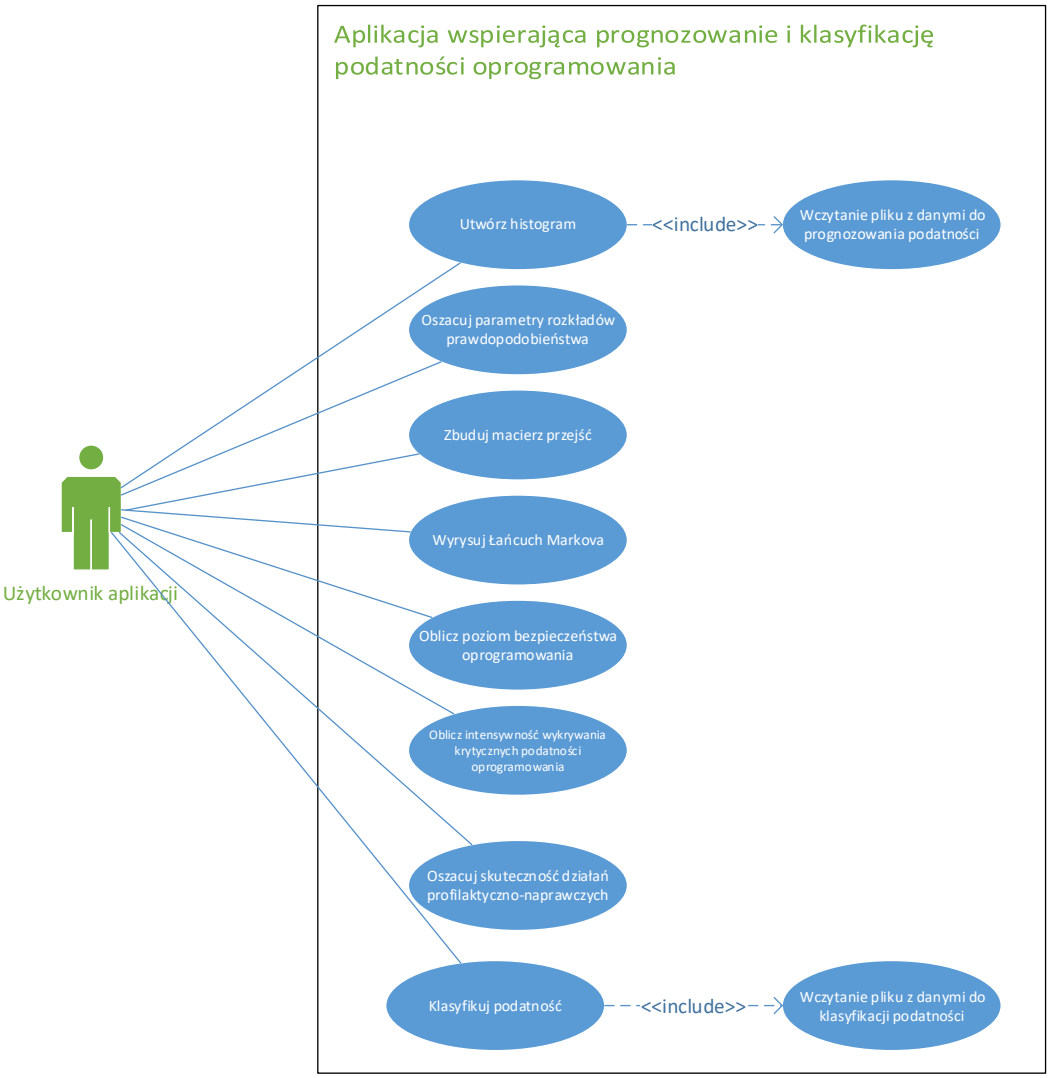


Rys. 61. Wykresy pudełkowe poszczególnych modeli

5. Autorskie środowisko obliczeniowe

5.1 Główne założenia

Główne funkcjonalności autorskiego prototypu aplikacji mają za zadanie realizować obliczenia w ustalonym procesie prognozowania podatności oprogramowania, w oparciu o zaproponowane metody w rozdziale 3 oraz umożliwić klasyfikację nowej podatności na podstawie zadanych parametrów i wybranego modelu klasyfikacji, opisanych w rozdziale 4. Poniżej przedstawiono diagram przypadków użycia dla budowanej aplikacji.



Rys. 62. Diagram przypadków użycia dla aplikacji wspierającej obliczenia

5.2 Szczegóły implementacji aplikacji

5.2.1 Wykorzystane technologie i narzędzia

Do implementacji prototypu aplikacji wykorzystano niżej wymienione oprogramowanie:

- **Microsoft Visual Studio 2022 Enterprise**– jest to zintegrowane środowisko do tworzenia oprogramowania firmy Microsoft, oferujące bogaty zestaw narzędzi z przyjaznym interfejsem graficznym użytkownika;
- **RStudio** – środowisko do tworzenia skryptów m.in. w języku R;
- **DevExpress WinForms 22.1** – pakiet narzędzi deweloperskich zastosowanych jako forma nakładki na środowisko Visual Studio 2022. Zawiera szeroki zakres narzędzi oraz dodatkowych kontrolek wykorzystywanych m.in. do budowy aplikacji okienkowych.

Interfejs oraz główne moduły aplikacji utworzone zostały w środowisku .Net Framework 4.8. Jednocześnie w celu realizacji obliczeń dla zaproponowanych metod i modeli wykorzystano możliwości środowiska do obliczeń i wizualizacji statystycznych R, poprzez utworzenie autorskich skryptów w języku R.

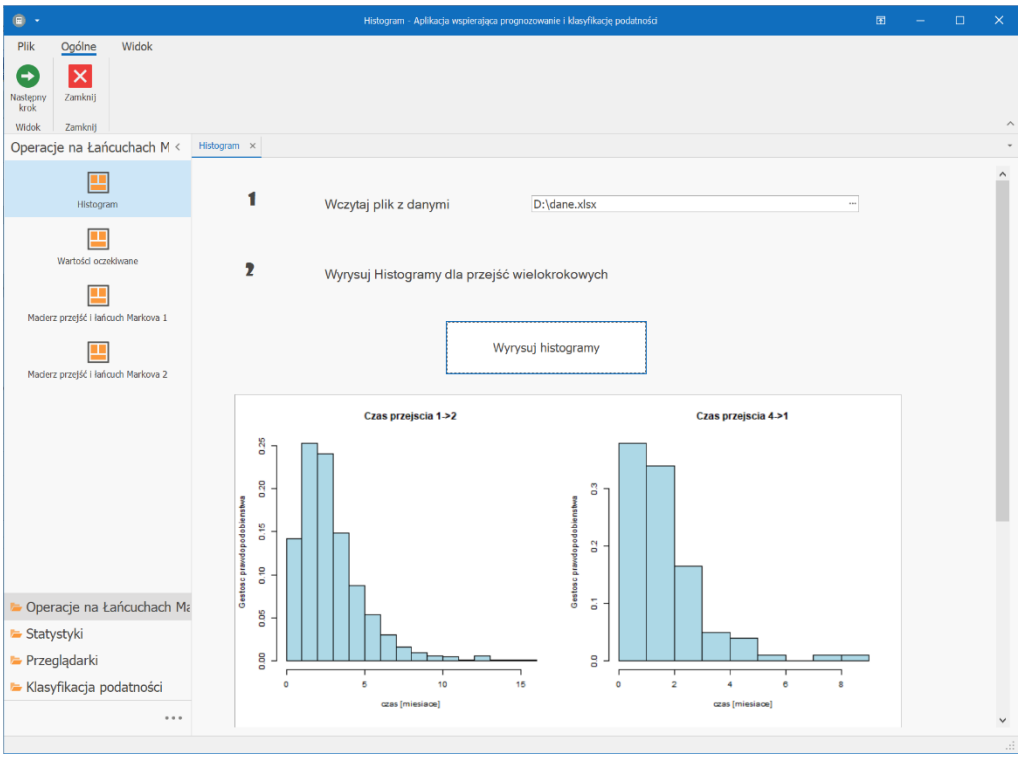
5.2.2 Interfejs graficzny oraz sposób działania aplikacji.

Interfejs wykonanego prototypu aplikacji wspierającej prognozowanie podatności oprogramowania przedstawia działanie głównych funkcjonalności w formie zakładek. Dzięki temu aplikacja staje się wygodna w eksploatacji, ponieważ wszystkie operacje wyświetlane są w jednym oknie, zbliżonym pod kątem wyglądu oraz rozmieszczenia elementów aplikacji do tego, który stosuje się w nowoczesnych aplikacjach desktopowych lub przeglądarkach internetowych.

Aplikacja posiada kolejno wymienione moduły:

- **Moduł *Histogram***

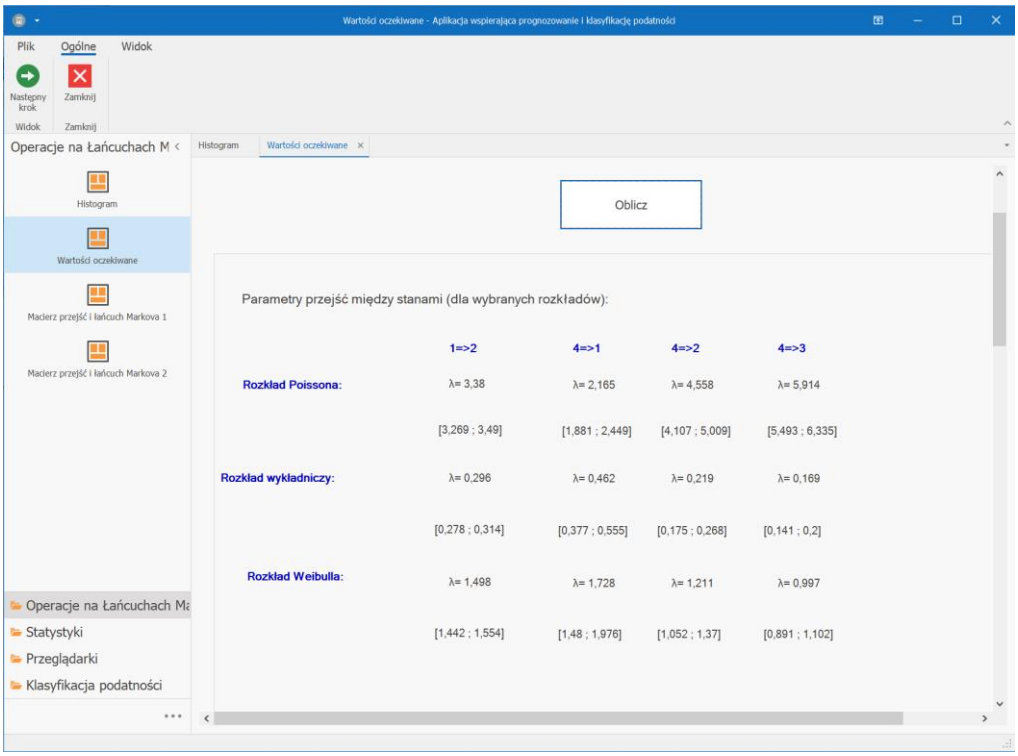
Główna funkcjonalność realizowana w module *Histogram* obejmuje wczytanie pliku zawierającego próbkę danych, wygenerowanie histogramów przy wykorzystaniu utworzonego skryptu w języku R oraz wyświetlenie histogramów przedstawiających rozkład empiryczny czasów przejścia między stanami (dla zaproponowanego łańcucha Markowa) z wczytanego pliku.



Rys. 63. Interfejs graficzny aplikacji - moduł *Histogram*

- **Moduł *Wartości oczekiwane***

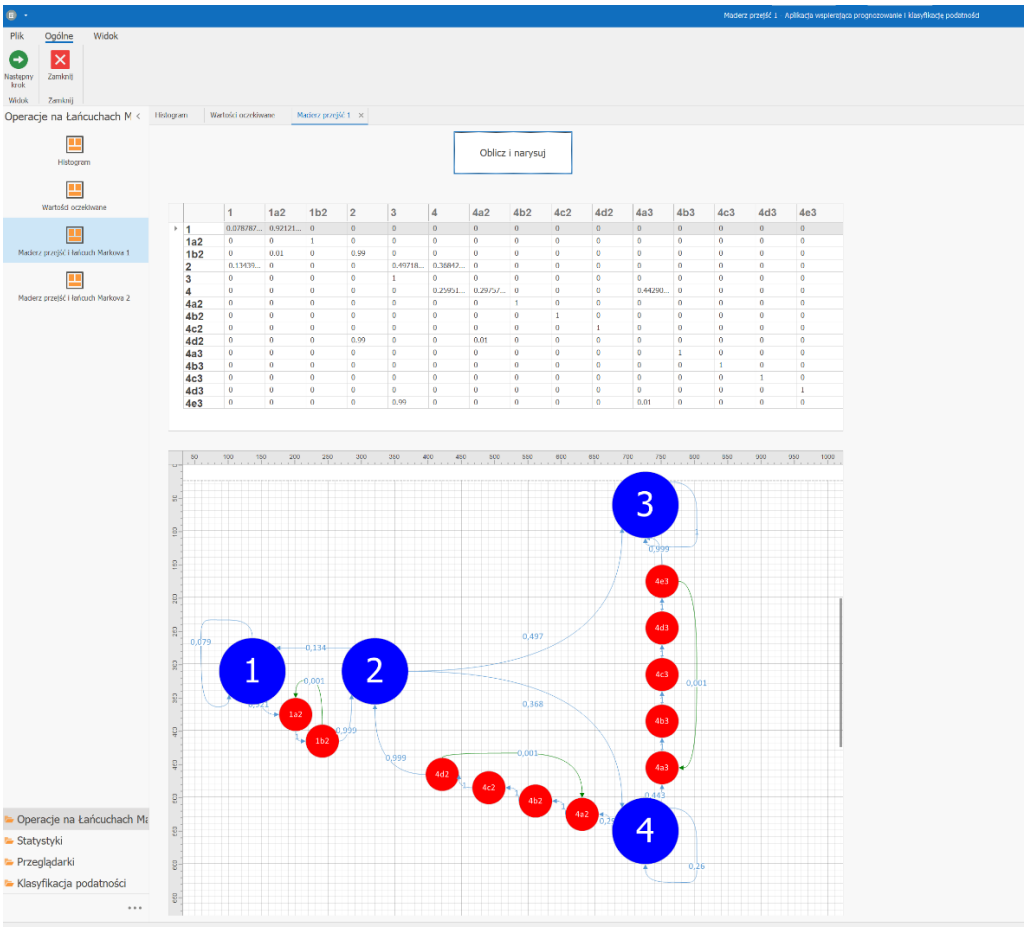
W module *Wartości oczekiwane* przy pomocy skryptu w języku R obliczane są parametry rozkładów prawdopodobieństwa i wartości oczekiwane przejść między stanami dla danych z wczytanego pliku wejściowego. W dalszej części wyświetlane są również histogramy wraz oszacowanymi i wyrysowanymi za pomocą kolejnego skryptu rozkładami Poissona, wykładniczym i Weibulla.



Rys. 64. Interfejs graficzny aplikacji - moduł *Wartości oczekiwane*

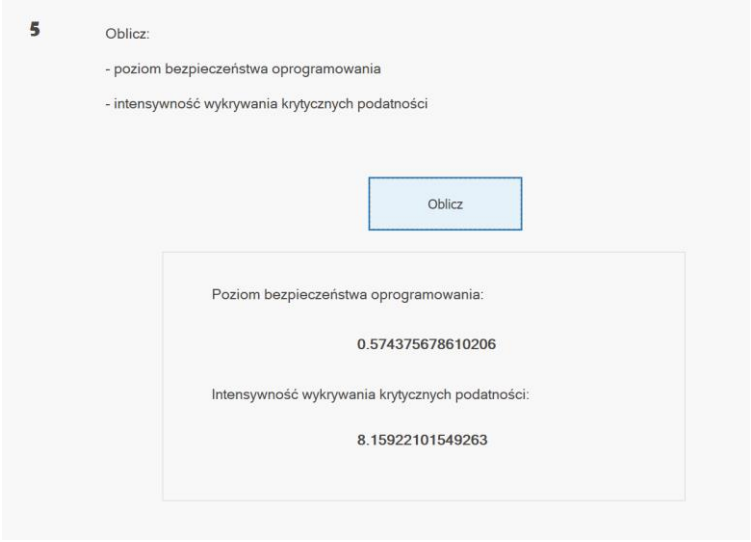
- **Moduł *Macierz przejścia bez uwzględnienia działań profilaktyczno-naprawczych i łańcuch Markowa***

W module *Macierz przejścia bez uwzględnienia działań profilaktyczno-naprawczych i łańcuch Markowa* realizowany jest kolejny etap obliczeń, gdzie dla oszacowanych wartości oczekiwanych rozkładów wygenerować możemy macierz przejść łańcucha Markowa dla modelu bez uwzględnienia działań profilaktyczno-naprawczych, z uwzględnieniem przejść krokowych. Macierz ta, jest generowana przy wykorzystaniu autorskiego skryptu, napisanego w języku R. Jednocześnie za pomocą wygenerowanej macierzy wyrysowywany jest w formie graficznej łańcuch Markowa dla tego przypadku.



Rys. 65. Interfejs graficzny aplikacji- moduł *Macierz przejścia bez uwzględnienia działań profilaktyczno-naprawczych i łańcuch Markowa* – generowanie macierzy przejścia oraz łańcucha Markowa

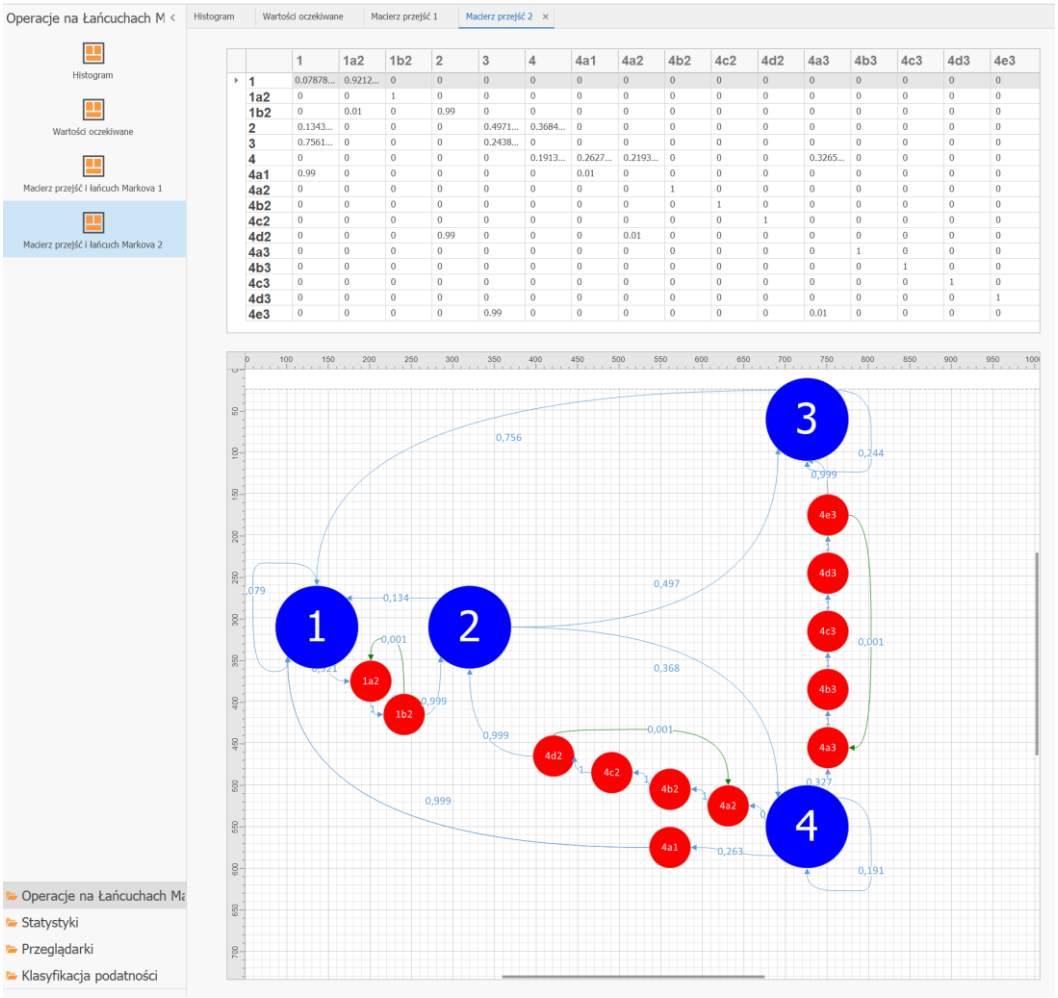
Następnie, kolejna funkcja aplikacji pozwala obliczyć poziom bezpieczeństwa oprogramowania oraz intensywność wykrywania krytycznych podatności oprogramowania.



Rys. 66. Interfejs graficzny aplikacji - *moduł Macierz przejścia bez uwzględnienia działań profilaktyczno-naprawczych i łańcuch Markowa* - obliczanie poziomu bezpieczeństwa oprogramowania oraz intensywności wykrywania krytycznych podatności

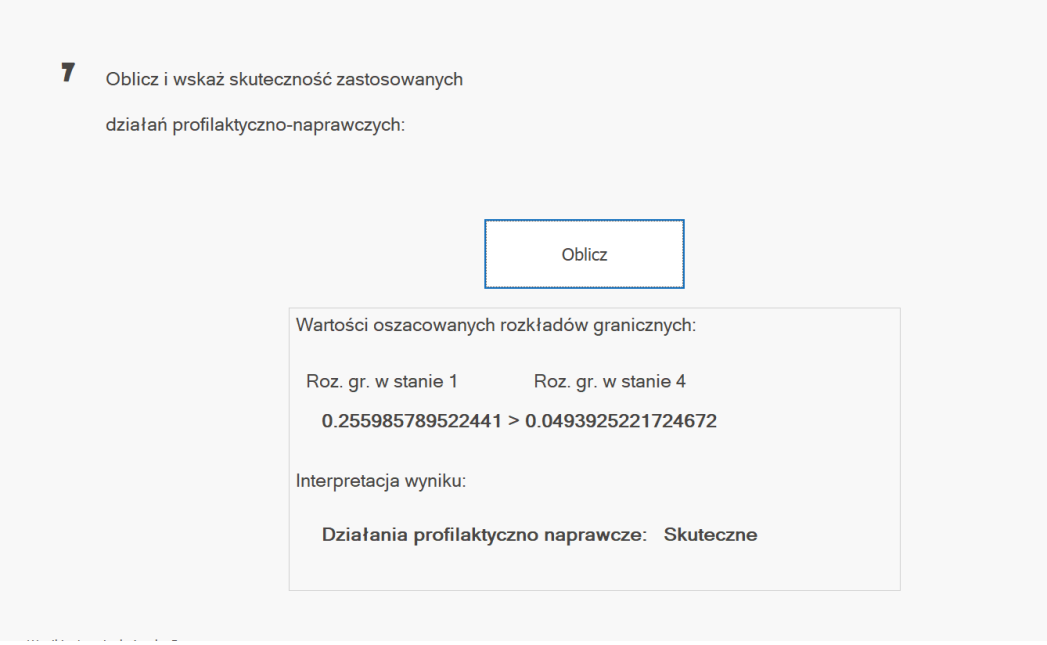
- **Moduł *Macierz przejścia z uwzględnieniem działań profilaktyczno-naprawczych i łańcuch Markowa***

Analogicznie jak w przypadku poprzednio opisywanego modułu, w module *Macierz przejścia z uwzględnieniem działań profilaktyczno-naprawczych i łańcuch Markowa* dla oszacowanych wartości oczekiwanych rozkładów wygenerować możemy macierz przejść łańcucha Markowa, tym razem dla modelu uwzględniającego działania profilaktyczno-naprawcze, z uwzględnieniem przejść krokowych. Macierz ta, jest generowana przy wykorzystaniu kolejnego autorskiego skryptu, napisanego w języku R. Jednocześnie za pomocą wygenerowanej macierzy wyrysowywany jest w formie graficznej łańcuch Markowa dla tego przypadku.



Rys. 67. ***Interfejs graficzny aplikacji- moduł Macierz przejścia z uwzględnieniem działań profilaktyczno-naprawczych i łańcuch Markowa*** **– generowanie macierzy przejścia oraz łańcucha Markowa**

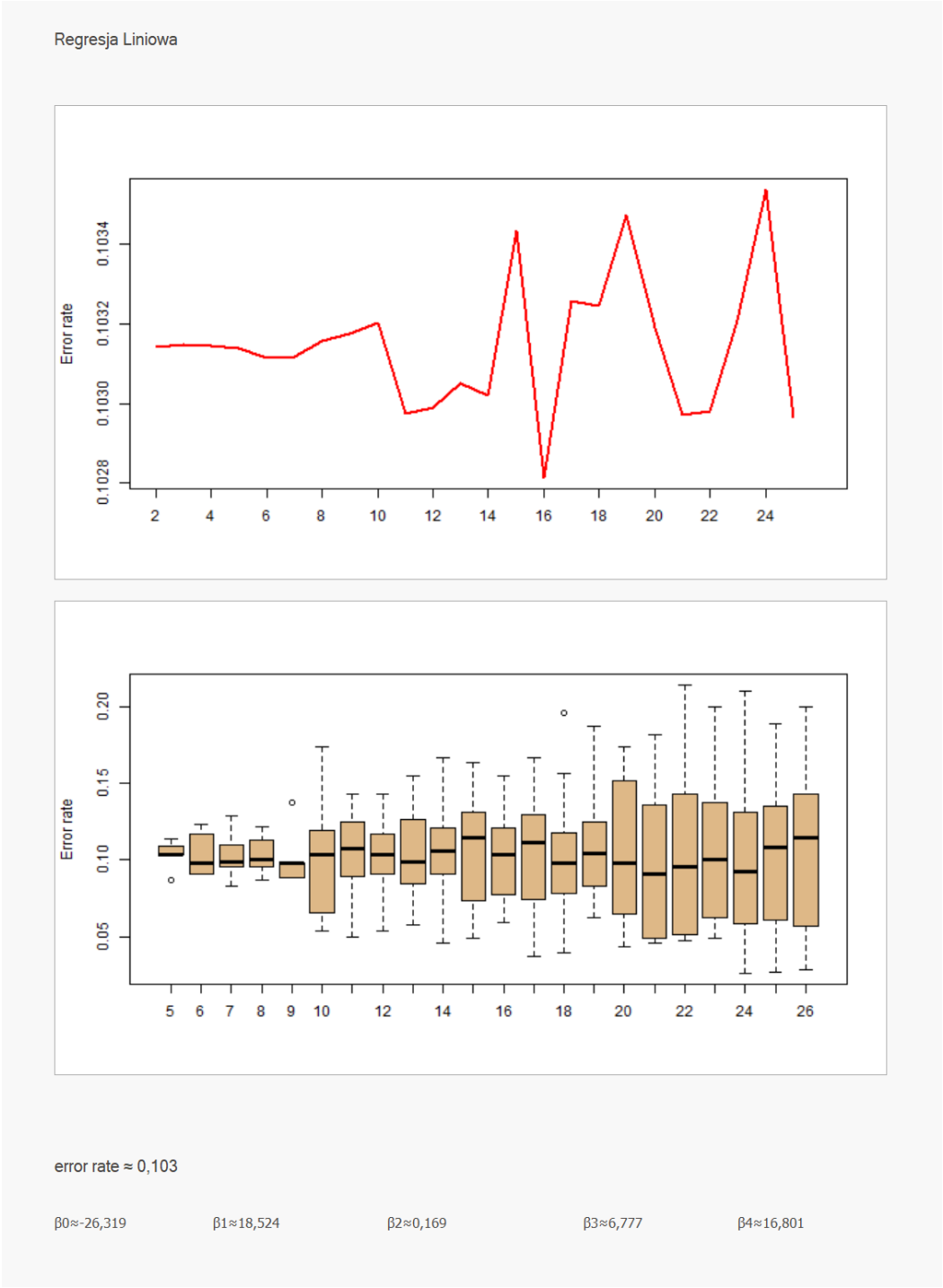
Następnie, kolejna funkcja pozwala obliczyć rozkłady graniczne w stanie 1 i 4 dla zadanego łańcucha Markowa, co finalnie pozwoli ocenić skuteczność zastosowanych działań profilaktyczno-naprawczych.



Rys. 68. Interfejs graficzny aplikacji- moduł *Macierz przejścia z uwzględnieniem działań profilaktyczno-naprawczych i łańcuch Markowa* – obliczanie skuteczności zastosowanych działań profilaktyczno-naprawczych

- **Moduł *Analiza danych do uczenia maszynowego***

Funkcjonalność realizowana w *module Analiza danych do uczenia maszynowego* obejmuje wczytanie pliku zawierającego próbkę danych do klasyfikacji podatności pochodzącą z wyodrębnionych danych z bazy danych podatności NVD, a następnie wygenerowanie i wyświetlenie wykresów wskaźników błędów oraz wykresy pudełkowe dla poszczególnych parametrów k w metodzie walidacji krzyżowej dla regresji liniowej, analizy dyskryminacji liniowej i analizy dyskryminacji kwadratowej. Realizowane jest to przy wykorzystaniu dedykowanych autorskich skryptów w języku R, których przykładowe wyniki wyświetlono na Rys. 69.

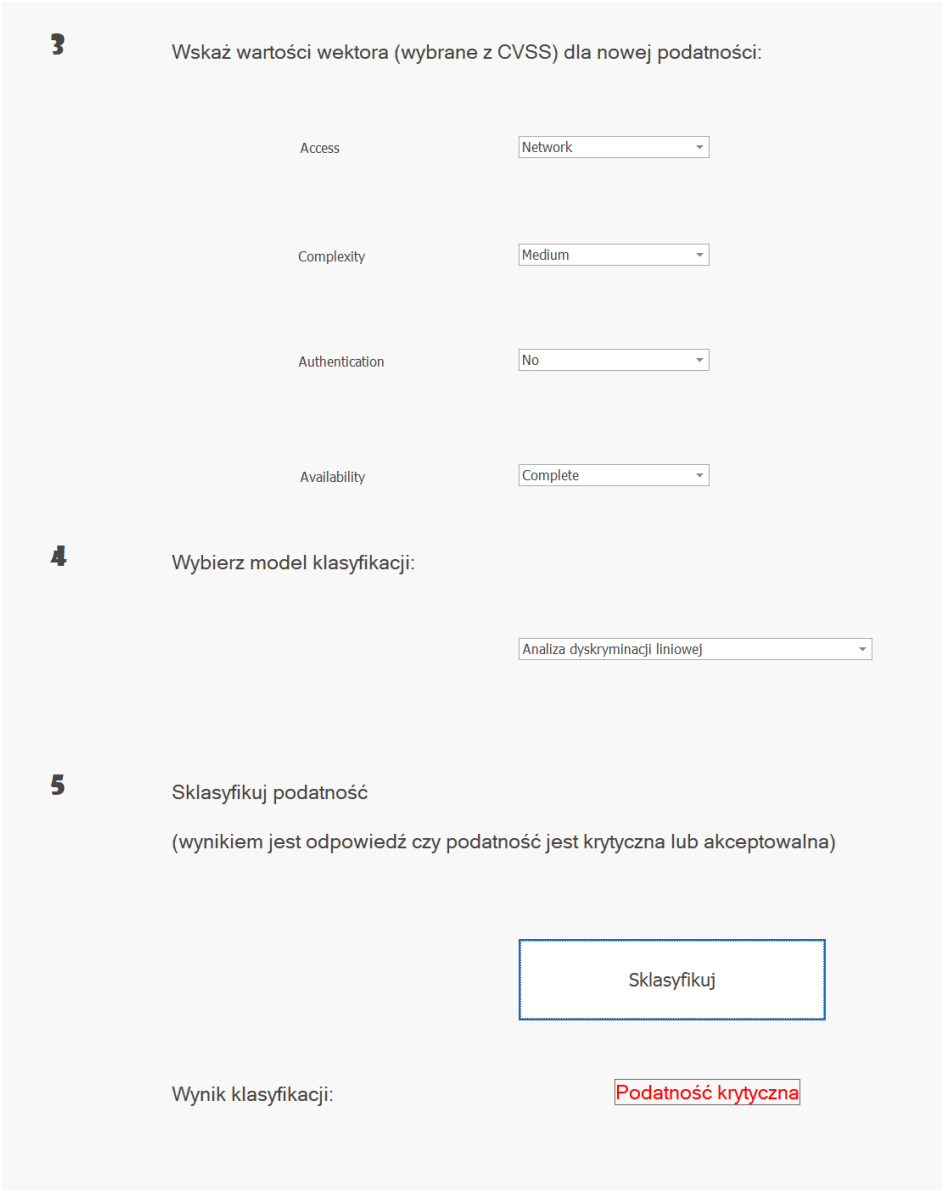


Rys. 69. Interfejs graficzny aplikacji – moduł *Analiza danych do uczenia maszynowego* – przykład wykresu wskaźników błędów oraz wykresu pudełkowe dla poszczególnych *k* w metodzie walidacji krzyżowej dla Regresji liniowej

- **Moduł *Klasyfikacja podatności***

Główną funkcją realizowaną przez moduł *Klasyfikacja podatności* jest wykonanie klasyfikacji dla nowo-zdefiniowanej podatności i przyporządkowanie do kategorii podatności akceptowalnych lub krytycznych. Użytkownik aplikacji wybiera wartości

składowych wektora dla nowej podatności oraz model do klasyfikacji, a następnie uruchamiając funkcję *Sklasyfikuj* następuje uruchomienie dedykowanego skryptu w języku R, który przeprowadza uczenie modelu, a następnie dla zadanych parametrów zwraca informacje czy podatność została sklasyfikowana jako akceptowalna lub krytyczna.



Rys. 70. Interfejs graficzny aplikacji - moduł *Klasyfikacja podatności*

Podsumowanie

W ostatnich latach podejmowanych zostało wiele prac nad zapewnieniem bezpieczeństwa oprogramowania, które w swoich założeniach uwzględniały obecność podatności oprogramowania. Podjęta tematyka niniejszej rozprawy doktorskiej wynika z braku dostatecznej liczby metod ilościowych do badania podatności oprogramowania, zaobserwowanego na etapie analizy literatury obszaru badawczego. Dodatkowo, przeprowadzona analiza statystyczna podatności oprogramowania udowadnia, iż podatności występują niemal w każdym dostępnym powszechnie oprogramowaniu. Jednocześnie, analiza dotychczasowych strategii cyberataków pokazuje, iż za fundament ich przeprowadzenia można uznać pomyślne wykorzystanie przez atakującego wykrytej na etapie rekonesansu podatności, m.in. podatności oprogramowania, podatności w konfiguracji sprzętowej czy podatności wynikającej z błędnego korzystania przez użytkownika z oprogramowania.

Rozpoznanie od strony technicznej potencjalnych podatności w oprogramowaniu niejednokrotnie stwarza wiele trudności w ustaleniu zależności analitycznych. Zadanie rozwiązania racjonalnego zapewnienia bezpieczeństwa oprogramowania implikuje zatem konieczność posiadania informacji prognostycznej dotyczącej potencjalnych podatności. Zastosowanie odpowiednio skonstruowanych modeli matematycznych w procesie wnioskowania ma na celu usprawnić jakościową i ilościową analizę zgromadzonych danych, a jednocześnie pozwoli na bardziej kompleksową i dokładniejszą ocenę poziomu jego bezpieczeństwa, wyznaczając tym samym ocenę możliwych przejść oprogramowania ze stanu bezpieczeństwa w stan podatny.

Warto zaznaczyć, że procesy prognozowania podatności oprogramowania oraz ocena jego bezpieczeństwa powinny być sobą nierozerwalnie związane. Informacje otrzymane w procesie predykcji podatności oprogramowania mogą być jednym z kluczowych elementów do realizacji procesu racjonalnego zarządzania bezpieczeństwem oprogramowania. Zarządzanie to polega na podjęciu określonych decyzji w kwestii dalszego przebiegu procesu projektowania, testowania lub utrzymania oprogramowania. Decyzje te mogą dotyczyć realizacji określonych zaleceń profilaktyczno-naprawczych stosowanych dla analizowanego oprogramowania.

Opracowanie rozprawy doktorskiej pt.: „*Model i metoda prognozowania podatności oprogramowania na zagrożenia w cyberprzestrzeni*” wymagało realizacji następujących zagadnień:

- rozpoznanie procesów i zjawisk charakterystycznych dla obszaru cyberbezpieczeństwa;
- przeprowadzenie analizy statystycznej i ilościową wykrytych podatności dla różnego rodzaju oprogramowania, zarejestrowanych w oficjalnych bazach danych podatności;
- dokonanie przeglądu metod analizy oraz prognozowania podatności oprogramowania dostępnych w literaturze;
- opracowanie autorskich modeli wykrywania podatności oprogramowania;
- opracowanie metody wyznaczania intensywności wykrywania krytycznych podatności oprogramowania;
- opracowanie metody oceny poziomu bezpieczeństwa oprogramowania z wykorzystaniem danych rejestrowanych na etapie projektowania, testowania oraz utrzymania oprogramowania;
- opracowanie metody oceniającej skuteczność działań profilaktyczno-naprawczych w zakresie eliminacji podatności w celu zapewnienia założonego poziomu bezpieczeństwa oprogramowania;
- weryfikację liczbową opracowanych metod.

Realizacja w/w przedsięwzięć oraz otrzymane w procesie badań wyniki jednoznacznie **potwierdzają** trafność sformułowanej tezy rozprawy doktorskiej:

Opracowanie modelu i metody prognozowania podatności oprogramowania na zagrożenia w cyberprzestrzeni pozwala na racjonalne kształtowanie poziomu bezpieczeństwa oprogramowania.

Z punktu widzenia nowatorstwa metodyki badań za najistotniejsze zatem w niniejszej rozprawie doktorskiej należy uznać zastosowanie modeli i metod probabilistycznych w procesie modelowania i oceny bezpieczeństwa oprogramowania. Zbudowane modele probabilistyczne działające na bazie rzeczywistych danych rejestrowanych na etapie projektowania, testowania oraz utrzymania oprogramowania pozwalają jednocześnie dokonać oceny zastosowanych działań profilaktyczno-naprawczych we wspomnianych etapach cyklu życia oprogramowania.

Wyniki pracy mogą zostać wykorzystane w następujących obszarach problemowych:

- możliwość prognozowania oceny bezpieczeństwa oprogramowania użytkowanego przez użytkownika końcowego oraz wewnątrz organizacji;
- możliwość oceny oraz automatyzacji i usprawnienia procesu projektowania potencjalnych działań profilaktyczno-naprawczych na potrzeby wytwarzania kolejnych produktów oprogramowania;
- możliwość udoskonalenia strategii zarządzania bezpieczeństwem informacji wewnątrz organizacji;
- możliwość udoskonalenia programów szkolenia z wytwarzania bezpiecznego oprogramowania (*SDL – Secure Development Lifecycle*) przeznaczonego dla zespołów projektowych.

Z poznawczego punktu widzenia rezultaty pracy pozwoliły na:

- statystyczną i ilościową analizę wykrytych podatności różnego rodzaju oprogramowania, zarejestrowanych m.in. w oficjalnych bazach danych podatności;
- poznanie sposobu działania oraz obszaru zastosowania istniejących modeli wykrywania podatności (*modele VDM*) oraz modeli predykcji podatności (*modele VPM*);
- doskonalenie metod predykcyjnych w aspektach prognozowania podatności i oceny bezpieczeństwa oprogramowania.

Jak zaznaczono wcześniej, wykorzystane w pracy modele i metody prognozowania podatności, z racji ograniczonego dostępu do rzetelnych danych rzeczywistych, mają zastosowanie przede wszystkim dla oprogramowania dedykowanego konkretnemu użytkownikowi końcowemu i takiego, które nie jest udostępniane do użytku zewnętrznego. Dalszy kierunek badań powinien zatem obejmować przypadki, w których oprogramowanie jest dostępne do nabycia dla ogółu użytkowników w sieci, niemniej jednak należy zaznaczyć, iż proces pozyskania rzetelnych danych do analizy może nie być w tym przypadku trywialny.

Bibliografia

[1] P. Zaskórski, Zarządzanie organizacją w warunkach ryzyka utraty informacyjnej ciągłości działania, Warszawa: WAT, 2011.

[2] A. Kari, „An Exceptional war That Ended in Victory for Estonia or an Ordinary e-Disturbance?", Estonian Narratives of The Cyber-Attacks in 2007, 2012.

[3] „Thanks, Stuxnet", New Scientist, 2011.

[4] „Duqu: son of Stuxnet", Computer Fraud & Security, 2011.

[5] K. Higgins, „Flame Gives Spyware A Next-Gen Update", Information Week, 2012.

[6] Zaufana Trzecia Strona, „Masowa, niezwykle skuteczna kampania ransomware wyłącza całe firmy", 2017. https://zaufanatrzeciastrona.pl/post/masowa-niezwykle-skuteczna-kampania-ransomware-wylacza-cale-firmy/.

[7] Gibson.W, Neuromancer, Warszawa: Wydawnictwo Książnica, 2009.

[8] P.Delvy, „Drugi Potop", Magazyn Sztuki, nr 13, 1997.

[9] Nowak.M, „Cybernetyczne przestępstwa - definicje i przepisy prawne", Cyberkłopoty i pułapki sieci, nr Nr 4/2010 (113), 2010.

[10] R. Kasprzyk, M. Paź i Z. Tarapata, „Modelowanie i symulacja cyberzagrożeń typu botnet", w XXI Warsztaty Naukowe PTSK, Białowieża, 2014.

[11] R. Hoffmann i B. Pacek, Działania Sił Zbrojnych w cyberprzestrzeni, Warszawa: Akademia Obrony Narodowej, 2013.

[12] Ministerstwo Cyfryzacji, Strategia cyberbezpieczeństwa Rzeczypospolitej Polskiej na lata 2017-2022, Warszawa, 2017.

[13] Ministerstwo Spraw Wewnętrznych i Administracji, „Rządowy program ochrony cyberprzestrzeni Rzeczypospolitej Polskiej na lata 2011-2016", Warszawa, 2010.

[14] K. Liderman, Analiza ryzyka i ochrona informacji w systemach komputerowych, Warszawa: Wydawnictwo Naukowe PWN, 2008.

[15] K. Liderman, Bezpieczeństwo informacyjne - nowe wyzwania, Warszawa: Wydawnictwo Naukowe PWN SA, 2017.

[16] CERT, „Katalog zagrożeń CERT.GOV.PL",http://www.cert.gov.pl/download/3/168/KatalogzagrozenCERTGOVPL.pdf. [Data uzyskania dostępu: 2 Październik 2016].

[17] K. Liderman, „Zarządzanie incydentami", w Przestępczość teleinformatyczna, Szczytno, WSPol, 2016, pp. 67-85.

[18] K. Liderman, „Dziesięć definicji", w Przestępczość teleinformatyczna 2017, Szczytno, WSPol, 2017, pp. 67-82.

[19] Rządowy Zespół Reagowania Na Incydenty Komputerowe, „Raport o stanie bezpieczeństwa cyberprzestrzeni RP w 2012 roku.", cert.gov.pl, Warszawa, 2013.

[20] Rządowy Zespół Reagowania Na Incydenty Komputerowe, „Raport o stanie bezpieczeństwa cybeprzestrzeni RP w 2013 roku", CERT.GOV.PL, Warszawa, 2014.

[21] Rządowy Zespół Reagowania Na Incydenty Komputerowe, „Raport o stanie bezpieczeństwa cyberprzestrzeni RP w 2014 roku", CERT.GOV.PL, Warszawa, 2015.

[22] Rządowy Zespół Reagowania Na Incydenty Komputerowe, „Raport o stanie bezpieczeństwa cyberprzestrzeni RP w 2015 roku", CERT.GOV.PL, Warszawa, 2016.

[23] ISO/IEC, „Norma ISO/IEC FIDIS 27000 - Information technology — Security techniques — Information Security management systems — Overview and vocabulary", 2009.

[24] Kaspersky Lab Global Research And Analysis Team (Great), „Kaspersky Security Bulletin 2013", Kaspersky Lab, 2014.

[25] Kaspersky Lab Global Research And Analysis Team (Great), „Kapserky Security Bulletin 2014", Kaspersky Lab, 2015.

[26] Kaspersky Lab Global Research And Analysis Team (Great), „Kaspersky Security Bulletin 2015", Kaspersky Lab, 2016.

[27] Kaspersky Lab Global Research And Analysis Team (Great), „Kaspersky Security Bulletin 2016", Kaspersky Lab, 2017.

[28] ISO/IEC, „Norma ISO/IEC FIDIS 27005:2008", 2008.

[29] „The Three Tenets of Cyber Security", http://www.spi.dod.mil/tenets.htm.

[30] A. Najgebauer, R. Antkiewicz, D. Pierzchała, T. Gutowski i P. Czuba, „Simulation Based Risk and Continuity of Services Analysis – A Case Study", w The 36th Annual European Simulation and Modelling Conference 2022, Porto, Portugal, 2022.

[31] K. Liderman, „O pomiarach bezpieczeństwa teleinformatycznego", Diagnostyka, tom nr 4 (40), pp. 113-118, 2006.

[32] R. Hoffman, „Stochastyczne modele cyklu życia podatności oprogramowania", Roczniki Kolegium Analiz Ekonomicznych, nr 49, pp. 273-285, 2018.

[33] S. Frei, B. Tellenbach i B. Plattner, „0-day patch - exposing vendors (in)security performance", w BlackHat Europe, Amsterdam, 2008.

[34] P. Mell i K. Scarfone, „A Complete Guide to the Common Vulnerability Scoring System", NIST, 2007.

[35] W. S. McPhee, „Operating System Integrity in OS/VS2.", IBM Sys. J., pp. 230-252, 1974.

[36] R. Abbott, „Security Analysis and Enhancements of Computer Operating Systems", Institute for Computer Science and Technology, National Bur. of Stnds, 1976.

[37] T. Aslam, „A taxonomy of Security Faults in the Unix Operating System", Purdue University, 1995.

[38] M. Bishop, „A Taxonomy of UNIX System and Network Vulnerabilities", Purdue, 1995.

[39] D. Lough, „A Taxonomy of Computer Attacks with Applications to Wireless Networks", Virginia Polytechnic Institute and State University, Virginia, 2001.

[40] F. Piessens, „A taxonomy of causes of software vulnerabilities in internet software”, w Proceedings of the 13th International Symposium on Software Reliability Engineering, 2002.

[41] A. Gray, „An Historical Perspective of Software Vulnerability Management”, 2003.

[42] V. Pothamsetty i B. Akyol, „A Vulnerability Taxonomy for Network Protocols: Corresponding Engineering Best”, w IASTED Int. Conf. on Commun., Internet, and Inform. Tech. (CIIT), Virgin Islands, 2004.

[43] R. Seacord, Secure Coding in C and C++, Addison-Wesley Professional, 2005.

[44] FORTIFY, „FortifySoftware”, www.fortifysoftware.com.

[45] A. Bazaz i A. James, „Towards a Taxonomy of Vulnerabilities”, w Proceedings of the 40th Annual Hawaii International Conference on System Sciences, 2007.

[46] V. Igure i R. Williams, „Taxonomies of Attacks and vulnerabilities in Computer Systems”, w IEEE Communications Surveys & Tutorials 1st Quarter 2008, 2008.

[47] MITRE, „Common Vulnerabilities and Exposures. The Standard for Information Security Vulnerability Names”.

[48] MITRE, „Common Configuration Enumeration - Standardized Identifiers for Security Configuration Issues and Exposures”, 2012.

[49] MITRE, „Scoring CWE's”, MITRE, 5 Wrzesień 2014. https://mitre.org/cwss/cwss_v.1.0.1.html.

[50] OSVDB, „OSVDB - Everything is Vulnerable”, https://blog.osvdb.org/. [Data uzyskania dostępu: 12 Listopad 2018].

[51] S. Özkan, „CVE Details - The ultimate security vulnerability datasource”, https://www.cvedetails.com/. [Data uzyskania dostępu: 7 Listopad 2018].

[52] Flashpoint, „VulnDB”, https://vulndb.flashpoint.io/.

[53] vuldb.com, „VULDB - the crowd-based vulnerability database”.

[54] P. Chen, L. Desmet i C. Huygens, „A Study on Advanced Persistent Threats”, Communications and Multimedia Security, pp. 63-72, 2014.

[55] T. Yadav i A. Rao, „Technical Aspects of Cyber Kill Chain”, Third International Symposium on Security in Computing and Communications, Sierpień 2015.

[56] O. Alhazmi, Y. Malaiya i I. Ray, „Measuring, analyzing and predicting security vulnerabilities in software systems”, Computers & Security, pp. 219-228, 2007.

[57] M. Dawson, D. N. Burrell, E. Rahim i S. Brewster, „Integrating Software Assurance into the Software Development Life Cycle (SDLC)”, JOURNAL OF INFORMATION SYSTEMS TECHNOLOGY & PLANNING, pp. 49-53, 2010.

[58] R. Kasprzyk i A. Stachurski, „A concept of standard-based vulnerability management automation for IT systems”, Computer Science and Mathematical Modelling 3, pp. 33-38, 2016.

[59] R. Kasprzyk i A. Stachurski, „Koncepcja procesu zarządzania podatnościami systemów teleinformatycznych”, w XXIII Warsztaty Naukowe PTSK, Zakopane, 2016.

[60] K. Scarfone i P. Mell, „The Common Configuration Scoring System (CCSS): Metrics for Software Security Configuration Vulnerabilities”, NIST, 2010.

[61] E. LeMay, K. Scarfone i P. Mell, „The Common Misuse Scoring System (CMSS): Metrics for Software Misuse Vulnerabilities”, NIST, 2012.

[62] H. Ghani, J. Luna i N. Suri, „Quantitative Assessment of Software Vulnerabilities Based on Economic-Driven Security Metrics”, w 2013 International Conference on Risks and Security of Internet and Systems (CRiSIS), La Rochelle, 2013.

[63] F. Innerhofer, „An empirically derived loss taxonomy based on publicly known security incidents”, w International Conference on Availability, Reliability and Security (ARES), Fukuoka, Japonia, 2009.

[64] M. Zieja, M. Zieja i A. Stachurski, „An outline of the method for predicting IT vulnerabilities”, w 22nd International Conference on Circuits, Systems, Communications and Computer, CSCC 2018, Majorka, 2018.

[65] M. Zieja, R. Kasprzyk, P. Migus i A. Stachurski, „A method for predicting IT vulnerabilities using branching processes - an outline”, w 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paryż, 2019.

[66] O. H. Alhazmi i Y. K. Malaiya, „Application of Vulnerability Discovery Models to Major Operating Systems”, IEEE TRANSACTIONS ON RELIABILITY, tom 1, Marzec 2008.

[67] E. Rescorla, „Is finding security holes a good idea?”, Security and Privacy, pp. 14-19, Styczeń-Luty 2005.

[68] A. Shrivastava, „Vulnerability Discovery Model for a Software System Using Stochastic Differential Equation”, w 1st International Conference on Futuristic trend in Computational Analysis and Knowledge Management (ABLAZE-2015), Nowe Delhi (Indie), 2015.

[69] R. Hoffmann, „Vulnerability Discovery Models for a Software System Using Stochastic Differential Equations”, Roczniki Kolegium Analiz Ekonomicznych / SGH, pp. 177-187 , 2017.

[70] J. HyunChul, K. Jinyoo i Y. K. Malaiya, „Vulnerability Discovery Modeling using Weibull Distribution”, w 19th International Symposium on Software Reliability Engineering, Redmond, 2008.

[71] A. Anand i N. Bhatt, „Vulnerability Discovery Modeling and Weighted Criteria Based Ranking”, The Indian Society for Probability and Statistics (ISPS) 2016, 2 Maj 2016.

[72] A. Goel i K. Okumoto, „Time-dependent error detection rate model for software and other performance measures”, IEEE Trans Reliab, pp. 206-211, 1979.

[73] M. Siavvas, E. Gelenbe, D. Kehagias i D. Tzovaras, „Static analysis-based approaches for Secure Software Development”, w Euro-CYBERSEC 2018: Security in Computer and Information Sciences, Londyn, 2018.

[74] P. Fatyga i R. Podraza, „Klasyfikacja danych - przegląd wybranych metod”, Zeszyty Naukowe Wydziału ETI Politechniki Gdańskiej. Technologie Informacyjne, pp. 55-60, 2010.

[75] M. Jimenez, M. Papadakis i Y. Traon, „Vulnerability Prediction Models : A case study on the Linux Kernel”, w 2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM), Raleigh, USA, 2016.

[76] V. H. Nguyen i T. L. M. S, „Predicting Vulnerable Software Components with Dependency Graphs”, w MetriSec '10 Proceedings of the 6th International Workshop on Security Measurements and Metrics, Bolzano, Włochy, 2010.

[77] Y. Shin, A. Meneely, L. Williams i J. Osborne, „Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Listopad 2011.

[78] S. Neuhaus, T. Zimmerman i A. Zeller, „Predicting Vulnerable Software Components”, w CCS '07 Proceedings of the 14th ACM conference on Computer and communications security , Alexandria, USA, 2007.

[79] R. Scandariato, J. Walden, A. Hovsepyan i W. Joosen, „Predicting Vulnerable Software Components via Text Mining”, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, pp. 993-1006, Październik 2014.

[80] Y. Pang, X. Xue i A. Namin, „Predicting Vulnerable Software Components through N-gram Analysis and Statistical Feature Selection”, 2015 IEEE 14th International Conference on Machine Learning and Applications, 2015.

[81] Y. Shin i L. Williams, „Can traditional fault prediction models be used for vulnerability prediction?”, Empirical Software Engineering, pp. 25-59, 09 Grudzień 2011.

[82] I. Guyon i A. Elisseeff, „An Introduction to Variable and Feature Selection”, Journal Machine Learning Research, pp. 1157-1182, 2003.

[83] T. Menzies, J. Greenwald i A. Frank, „Data Mining Static Code Attributes to Learn Defect Predictors”, IEEE Transactions on Software Engineering, pp. 2-13, 2007.

[84] Ł. Borchmann, „Wyrazowe modele języka, n-gramy i SRILM”, http://www.borchmann.pl/dydaktyka/wyrazowe-modele-jezyka-n-gramy-i-srilm. [Data uzyskania dostępu: 23 Październik 2018].

[85] B. Ziółko i D. Skórzok, N-grams model for Polish, InTech Publisher, 2011.

[86] P. Bremaud, Probability Theory and Stochastic Processes, Springer Nature Switzerland, 2020.

[87] J. Koronacki i J. Mielniczuk, Statystyka dla studentów kierunków technicznych i przyrodniczych, Warszawa: Wydawnictwa Naukowo-Techniczne, 2006.

[88] J. Norris, Markov Chains - Cambridge Series in Statistical and Probabilistic Mathematics, Cambridge University Press, 1997.

[89] O. Haggstrom, Finite Markov Chains And Algorithmic Applications, Cambridge: Camridge University Press, 2002.

[90] N. Privault, Understanging Markov Chains. Examples and Applications, Singapur: Springer, 2013.

[91] C. Bishop, Pattern Recognition and Machine Learning, 2006 Springer Science+Business Media, LLC, 2006.

Wykaz tabel

Wykaz rysunków

Załącznik 1

Podstawowe definicje i własności z probabilistyki [86] [87] [88] [89] [90].

Przestrzeń prób Ω to zbiór wszystkich możliwych wyników $\omega \in \Omega$ pewnego losowego eksperymentu. Powiemy, że $\mathcal{F} \in 2^{\Omega}$ (przez 2^{Ω} rozumiemy rodzinę wszystkich podzbiorów zbioru Ω) jest **σ-algebrą** (lub **σ-ciałem**}) jeśli:

- $\Omega \in \mathcal{F}$.
- Jeśli $A \in \mathcal{F}$, to $A^c \in \mathcal{F}$ (gdzie $A^c = \Omega \backslash A$).
- Jeśli $A_i \in \mathcal{F}$ dla $i = 1,2,3, \dots$, to $\bigcup_i A_i \in \mathcal{F}$.

W celu wprowadzenia precyzyjnej definicji zmiennej losowej, konieczne jest wprowadzenie pojęcia przestrzeni topologicznej. Niech dany będzie zbiór X i niech τ będzie rodziną podzbiorów zawartych w X i spełniającą następujące warunki:

- $X \in \tau, \emptyset \in \tau$,
- jeśli $U, V \in \tau$, to $U \cap V \in \tau$,
- jeśli $G_i \in \tau$ dla $i = 1,2, \dots, n$, to $\bigcap_{i=1}^{n} G_i \in \tau$.

Wówczas przestrzeń X z tak zdefiniowaną rodzinę podzbiorów τ nazywamy **przestrzenią topologiczną**, natomiast elementy rodziny τ nazywamy **zbiorami otwartymi**.

Niech dana będzie dowolna kolekcja $\mathcal{C}$ podzbiorów Ω, rozważmy wszystkie σ-algebry zawierające rodzinę $\mathcal{C}$ (istnieje co najmniej jedna taka σ-algebra, mianowicie 2^{Ω}). Biorąc przecięcie mnogościowe wszystkich takich σ-algebr otrzymamy najmniejszą σ-algebrę zawierającą $\mathcal{C}$, którą nazywamy **σ-algebrą generowaną przez $\mathcal{C}$**. Jeśli Ω jest przestrzenią topologiczną, wtedy σ-algebra generowana przez kolekcję wszystkich zbiorów otwartych w Ω nazywa się **Borelowską σ-algebrą na Ω** i jest oznaczana przez $\mathcal{B}(\Omega)$. Elementy borelowskiej σ-algebry nazywamy **zbiorami borelowskimi**.

Jednym z bardziej fundamentalnych wyników dotyczących Borelowskich σ-algebr jest fakt, że $\mathcal{B}(\mathbb{R})$ jest generowany przez przedziały postaci $(-\infty, a]$, gdzie $a \in \mathbb{Q}$.

Przez **zdarzenie** będziemy rozumieli kolekcję wyników, które są reprezentowane prze podzbiory Ω. Parę $(\Omega, \mathcal{F})$, gdzie $\mathcal{F}$ jest σ-algebrą podzbiorów Ω, nazywamy **przestrzenią mierzalną**. Niech $(\Omega, \mathcal{F})$ będzie przestrzenią mierzalną, miarą na tej przestrzeni nazwiemy funkcję $\mu: \mathcal{F} \to [0, \infty]$ mającą następujące własności:

- $\mu(A) \geq \mu(\emptyset) = 0$ dla każdego $A \in \mathcal{F}$.

- $\mu(\bigcup_n A_n) = \sum_n \mu(A_n)$ dla każdej przeliczalnej kolekcji parami rozłącznych zbiorów $A_n \in \mathcal{F}$.

Ponadto, jeśli $\mu(\Omega) = 1$, wówczas miarę μ będziemy nazywali **prawdopodobieństwem**, które będziemy oznaczali przez $\mathbb{P}$.

Przestrzenią z miarą nazwiemy trójkę $(\Omega, \mathcal{F}, \mu)$, gdzie μ jest miarą na przestrzeni mierzalnej $(\Omega, \mathcal{F})$. Przestrzeń z miarą $(\Omega, \mathcal{F}, \mathbb{P})$ z prawdopodobieństwem $\mathbb{P}$ nazwiemy **przestrzenią probabilistyczną**. σ-algebra $\mathcal{F}$ zawsze zawiera co najmniej zbiór Ω i jego dopełnienie, zbiór pusty $\emptyset$. Koniecznie mamy : $\mathbb{P}(\Omega) = 1$ oraz $\mathbb{P}(\emptyset) = 0$. Można pokazać, że istnieje jednoznaczna miara μ na przestrzeni $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ określona wzorem $\mu(a, b) = b - a$, którą będziemy nazywali **miarą Lebesgue'a**.

Niech $(\Omega_1, \mathcal{F}_1)$ oraz $(\Omega_2, \mathcal{F}_2)$ będą przestrzeniami mierzalnymi, wówczas odwzorowanie $f: \Omega_1 \to \Omega_2$ nazywamy **mierzalnym** jeśli $f^{-1}(A) \in \mathcal{F}_1$, dla każdego $A \in \mathcal{F}_2$. Niech $(\Omega, \mathcal{F}, \mathbb{P})$ będzie przestrzenią probabilistyczną. Odwzorowanie mierzalne $X: \Omega \to \mathbb{R}$ z przestrzeni mierzalnej $(\Omega, \mathcal{F})$ w przestrzeń mierzalną $(\mathbb{R}, \mathcal{B}(\mathbb{R}))$ nazywamy **zmienną losową**.

Dla danej zmiennej losowej i zbioru Borelowskiego A przez $\{X \in A\}$ będziemy oznaczali zdarzenie

$$\{X \in A\} = \{\omega \in \Omega: X(\omega) \in A\}.$$

Niech dane teraz będą dwa zdarzenia $A, B \subset \Omega$, takie, że $\mathbb{P}(B) \neq 0$. Prawdopodobieństwo:

$$\mathbb{P}(A|B) \coloneqq \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

nazywamy **prawdopodobieństwem warunkowy zajścia zdarzenie A pod warunkiem zdarzenia B**. Zauważmy, że jeśli $\mathbb{P}(B) = 1$, wówczas mamy $\mathbb{P}(A \cap B^c) \leq \mathbb{P}(B^c)$, zatem $\mathbb{P}(A \cap B^c) = 0$, co implikuje

$$\mathbb{P}(A) = \mathbb{P}(A \cap B) + \mathbb{P}(A \cap B^c) = \mathbb{P}(A \cap B)$$

oraz $\mathbb{P}(A|B) = \mathbb{P}(A)$.

Przypomnijmy również następującą własność:

$$\mathbb{P}\left(B \cap \bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} \mathbb{P}(B \cap A_n) = \sum_{n=1}^{\infty} \mathbb{P}(B|A_n)p(A_n) = \sum_{n=1}^{\infty} \mathbb{P}(A_n|B)p(B),$$

dla dowolnej rodziny parami rozłącznych zdarzeń $(A_n)_{n\geq 1}$ oraz dla $\mathbb{P}(B) > 0$. To również pokazuje, że prawdopodobieństwo warunkowe jest prawdopodobieństwem (w rozumieniu miary), w sensie, że jeśli $\mathbb{P}(B) > 0$, to

- $\mathbb{P}(\Omega|B) = 1$ oraz
- $\mathbb{P}(\bigcup_{n=1}^{\infty} A_n|B) = \sum_{n=1}^{\infty} \mathbb{P}(A_n|B)$, jeśli $A_k \cap A_l = \emptyset$, $k \neq l$.

W szczególności, jeśli $\bigcup_{i=1}^{\infty} A_n = \Omega$, $(A_n)_{n\geq 1}$ staje się **podziałem zbioru Ω** i otrzymujemy **wzór na prawdopodobieństwo całkowite**:

$$\mathbb{P}(B) = \sum_{n=1}^{\infty} \mathbb{P}(B \cap A_n) = \sum_{n=1}^{\infty} \mathbb{P}(A_n|B)\mathbb{P}(B) = \sum_{n=1}^{\infty} \mathbb{P}(B|A_n)\mathbb{P}(A_n),$$

przy założeniu, że $A_i \cap A_j \neq \emptyset$, $i \neq j$, $\mathbb{P}(B) > 0$ oraz $n \geq 1$.

Powiemy, że zdarzenia $A, B \subset \Omega$ są **niezależne** jeśli

$$\mathbb{P}(A|B) = \mathbb{P}(A),$$

co jest równoważne warunkowi

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B).$$

Rozkładem prawdopodobieństwa zmiennej losowej $X: \Omega \to \mathbb{R}$ nazywamy kolekcję zbiorów:

$$\{\mathbb{P}(X \in A): A \in \mathcal{B}(\mathbb{R})\}.$$

W rzeczywistości rozkład zmiennej losowej X sprowadza się do znajomości jednej z poniższych kolekcji:

$$\{\mathbb{P}(a < X \leq b): a < b \in \mathbb{R}\}, \quad \{\mathbb{P}(X \leq a): a \in \mathbb{R}\}, \quad \{\mathbb{P}(X \geq a): a \in \mathbb{R}\}.$$

W przypadku gdy rozkład dopuszcza tzw. gęstość, rozkład zmiennej losowej X dany jest przez

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f(x)dx$$

gdzie funkcja $f: \mathbb{R} \to \mathbb{R}_+$ nazywa się gęstością rozkładu zmiennej losowej X. W szczególności mamy

$$\int_{-\infty}^{+\infty} f(x)dx = \mathbb{P}(-\infty \leq X \leq +\infty) = 1$$

dla każdej funkcji gęstości $f: \mathbb{R} \to \mathbb{R}_+$. Funkcja gęstości f_X może być odtworzona z funkcji rozkładu tzn.

$$x \longmapsto \mathbb{P}(X \leq x) = \int_{-\infty}^{x} f_X(s)ds\,, \quad x \in \mathbb{R},$$

oraz

$$x \longmapsto \mathbb{P}(X \geq x) = \int_{x}^{+\infty} f_X(s)ds\,, \quad x \in \mathbb{R},$$

jako, że

$$f_X(x) = \frac{\partial}{\partial x}\int_{-\infty}^{x} f_X(s)ds = -\frac{\partial}{\partial x}\int_{x}^{+\infty} f_X(s)ds\,, \quad x \in \mathbb{R}.$$

Zauważmy jeszcze, że w przypadku rozkładu dopuszczającego gęstość, dla każdego $a \in \mathbb{R}$ mamy $\mathbb{P}(X = a) = \int_a^a f(x)dx = 0$. W dalszej części pracy przez rozkład zmiennej losowej X, będziemy rozumieli funkcję określona wzorem $F(x) = \mathbb{P}(X < x), x \in \mathbb{R}$

Niech $f_X: \mathbb{R} \to \mathbb{R}_+$ będzie gęstością zmiennej losowej, wówczas przez **wartość oczekiwaną** rozumiemy

$$\mathbb{E}[X] = \int_{-\infty}^{+\infty} x f_X(x)dx,$$

lub w bardziej ogólnej postaci

$$\mathbb{E}\{\Psi(x)\} = \int_{-\infty}^{+\infty} \Psi(x) f_X(x)dx,$$

gdzie Ψ jest funkcją całkowalną.

W przypadku gdy rozkład nie dopuszcza gęstości, tzn. rozkład jest dyskretny, zamiast pojęcia gęstości rozważamy pojęcie funkcji masy. Wówczas rozkład zmiennej losowej X dany jest przez

$$\mathbb{P}(a \leq X \leq b) = \sum_{x=a}^{b} f_X(x)$$

gdzie funkcja $f: \mathbb{Z} \to \mathbb{R}_+$ nazywa się funkcją masy zmiennej losowej X i jest określona wzorem $f_X(x) = \mathbb{P}(X = x)$. W szczególności mamy

$$\sum_x f_X(x) = 1$$

Niech $f_X: \mathbb{Z} \to \mathbb{R}_+$ będzie funkcją masy zmiennej losowej, wówczas analogicznie do przypadku ciągłego (dopuszczającym gęstość prawdopodobieństwa) rozważamy wartość oczekiwaną$\mathbb{E}[X] = \sum_x x f_X(x)$.

Załącznik 2

Przykładowy podzbiór wykorzystanej próbki danych z testowania, przedstawiający uśrednione wartości czasów przejść między stanami dla zaproponowanych łańcuchów Markova z rysunków Rys. 29 i Rys. 31:

id	rel_ver	resp_ent	comm_test	lght_vul_fix	re_test	re_class	allow_vuln	cr_vuln_fix	crit_abort	no_vuln_det
5FE2879D-	1.0.0.2	entHMS/mber	2		6		1			
122E2C43-	1.1.0.0	entHMS/mber	6					1		
56CC7612-	1.2.0.1	entHMS/mber	2	1			1			
CE8AFB93	1.4.0.2	vbis/lcosp23	4					1		
661B1089-	12.0.0.0	entHMS/mber	3		8		1			1
C3DCC0A6	2.1.0.1	entHMS/mber	2	4	5		1			
0455B444-	2.2.0.1	vbis/azer01	6	1			1			
0A01D82A	2.2.0.1	vbis/lcosp23	2		7		1			
9FB402AA	2.2.0.1	entHMS/mber	1	1			1			
0269A2BA	2.3.0.0	entHMS/mber	6	2			1			
BC1A37E6	2.4.1.0	entHMS/mber	1							1
7430645D	3.0.0.0	vbis/azer01	2					1		
B8D909A4	3.0.0.0	vbis/lcosp23	3					1		
C765DE52	3.0.0.0	entHMS/mber	3	1	6		1	1		
138A3B91	3.1.0.0	vbis/lcosp23	5					1		
92150E77-	3.1.0.0	entHMS/mber	7	2			1			
9662C1EF-	3.1.1.0	entHMS/mber	1							1
A5706E07	3.1.1.0	vbis/lcosp23	4	3			1			
C243D598	3.1.1.0	tst/dom01rb	4							1
D91408E1	3.1.1.0	vbis/azer01	6	5	3		1			
089A8CE9	3.1.1.1	vbis/lcosp23	6	1			1			
637D100B	3.1.1.1	tst/dom01rb	5		2					1
8A6A7F86	3.1.1.1	entHMS/mber	2			5		1		
5DBCE111	3.2.1.0	entHMS/mber	1	2			1			
72066B31-	3.2.1.0	tst/dom01rb	3	2			1			
8C1BEBB0	3.2.1.0	vbis/azer01	4							1
A856CAD7	3.8.1.0	vbis/lcosp23	4							1
C8759D0B	3.9.1.1	vbis/lcosp23	2					1		
F4975191-	3.9.1.2	vbis/lcosp23	2			8		1		
B4B4FB48	4.0.0.1	entHMS/mber	1	2			1			
41DA19A1	4.0.0.2	entHMS/mber	3							1
6E3A5008	4.0.1.0	entHMS/mber	4					1		
072F3FC3-	4.0.1.1	tst/dom01rb	4					1		
99FA8F6F-	4.0.1.1	vbis/azer01	2	2			1			
9DF5C69C	5.1.0.0	entHMS/mber	7							1
7F256E7A-	6.2.0.0	entHMS/mber	2		1		1			
3CA4F469	7.0.0.0	vbis/lcosp23	3							1
6E515BFC-	9.0.0.0	vbis/lcosp23	2	3			1			

Załącznik 3

Przykładowy podzbiór rekordów z bazy danych podatności *NVD* wykorzystywanych do uczenia modeli w celu klasyfikacji podatności - widok przeglądarki podatności w autorskiej aplikacji:

ID	Problem...	CWE	Publish D...	Score	Exploitability Score	Impact Score	Vector
CVE-2019-9904	CWE-400	**Uncontrolled Resource Consu...**	21.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:N/A:P
CVE-2019-9908	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9909	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9910	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9911	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9912	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9913	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9914	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9915	CWE-601	**URL Redirection to Untrusted ...**	22.03.2019	5,8	8,6	4,9	AV:N/AC:M/Au:N/C:P/I:P/A:N
CVE-2019-9917	CWE-20	**Improper Input Validation**	27.03.2019	4	8	2,9	AV:N/AC:L/Au:S/C:N/I:N/A:P
CVE-2019-9918	CWE-89	**Improper Neutralization of Sp...**	29.03.2019	6,4	10	4,9	AV:N/AC:L/Au:N/C:P/I:P/A:N
CVE-2019-9919	CWE-79	**Improper Neutralization of In...**	29.03.2019	3,5	6,8	2,9	AV:N/AC:M/Au:S/C:N/I:P/A:N
CVE-2019-9921	CWE-284	**Improper Access Control**	29.03.2019	4	8	2,9	AV:N/AC:L/Au:S/C:P/I:N/A:N
CVE-2019-9922	CWE-22	**Improper Limitation of a Path...**	29.03.2019	5	10	2,9	AV:N/AC:L/Au:N/C:P/I:N/A:N
CVE-2019-9923	CWE-476	**NULL Pointer Dereference**	22.03.2019	5	10	2,9	AV:N/AC:L/Au:N/C:N/I:N/A:P
CVE-2019-9924	CWE-20	**Improper Input Validation**	22.03.2019	7,2	3,9	10	AV:L/AC:L/Au:N/C:C/I:C/A:C
CVE-2019-9925	CWE-79	**Improper Neutralization of In...**	22.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9927	CWE-20	**Improper Input Validation**	22.03.2019	7,5	10	6,4	AV:N/AC:L/Au:N/C:P/I:P/A:P
CVE-2019-9928	CWE-119	**Improper Restriction of Opera...**	24.04.2019	6,8	8,6	6,4	AV:N/AC:M/Au:N/C:P/I:P/A:P
CVE-2019-9930	CWE-190	**Integer Overflow or Wraparound**	29.08.2019	10	10	10	AV:N/AC:L/Au:N/C:C/I:C/A:C
CVE-2019-9931	CWE-20	**Improper Input Validation**	29.08.2019	7,8	10	6,9	AV:N/AC:L/Au:N/C:N/I:N/A:C
CVE-2019-9932	CWE-119	**Improper Restriction of Opera...**	29.08.2019	10	10	10	AV:N/AC:L/Au:N/C:C/I:C/A:C
CVE-2019-9933	CWE-119	**Improper Restriction of Opera...**	29.08.2019	10	10	10	AV:N/AC:L/Au:N/C:C/I:C/A:C
CVE-2019-9934	CWE-284	**Improper Access Control**	28.08.2019	5	10	2,9	AV:N/AC:L/Au:N/C:P/I:N/A:N
CVE-2019-9935	CWE-284	**Improper Access Control**	28.08.2019	5	10	2,9	AV:N/AC:L/Au:N/C:P/I:N/A:N
CVE-2019-9936	CWE-125	**Out-of-bounds Read**	22.03.2019	5	10	2,9	AV:N/AC:L/Au:N/C:P/I:N/A:N
CVE-2019-9937	CWE-476	**NULL Pointer Dereference**	22.03.2019	5	10	2,9	AV:N/AC:L/Au:N/C:N/I:N/A:P
CVE-2019-9938	CWE-200	**Exposure of Sensitive Informa...**	22.03.2019	2,9	5,5	2,9	AV:A/AC:M/Au:N/C:P/I:N/A:N
CVE-2019-9939	CWE-287	**Improper Authentication**	22.03.2019	5,8	6,5	6,4	AV:A/AC:L/Au:N/C:P/I:P/A:P
CVE-2019-9942	CWE-200	**Exposure of Sensitive Informa...**	23.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:P/I:N/A:N
CVE-2019-9945	CWE-77	**Improper Neutralization of Sp...**	23.03.2019	10	10	10	AV:N/AC:L/Au:N/C:C/I:C/A:C
CVE-2019-9947	CWE-93	**Improper Neutralization of CR...**	23.03.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9949	CWE-59	**Improper Link Resolution Befo...**	23.05.2019	9	8	10	AV:N/AC:L/Au:S/C:C/I:C/A:C
CVE-2019-9950	CWE-287	**Improper Authentication**	24.04.2019	7,5	10	6,4	AV:N/AC:L/Au:N/C:P/I:P/A:P
CVE-2019-9951	CWE-434	**Unrestricted Upload of File wit...**	24.04.2019	7,5	10	6,4	AV:N/AC:L/Au:N/C:P/I:P/A:P
CVE-2019-9955	CWE-79	**Improper Neutralization of In...**	22.04.2019	4,3	8,6	2,9	AV:N/AC:M/Au:N/C:N/I:P/A:N
CVE-2019-9956	CWE-119	**Improper Restriction of Opera...**	24.03.2019	6,8	8,6	6,4	AV:N/AC:M/Au:N/C:P/I:P/A:P
CVE-2019-9957	CWE-79	**Improper Neutralization of In...**	24.06.2019	3,5	6,8	2,9	AV:N/AC:M/Au:S/C:N/I:P/A:N
CVE-2019-9958	CWE-352	**Cross-Site Request Forgery (C...**	24.06.2019	6,8	8,6	6,4	AV:N/AC:M/Au:N/C:P/I:P/A:P