POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Inżynierii Oprogramowania

Inżynieria oprogramowania, procesów biznesowych i baz danych

Marcin Węgłowski
Nr albumu S15237

Generyczny skaner bezpieczeństwa dla systemów internetowych

Praca magisterska

Dr inż. Mariusz Trzaska

Warszawa, Wrzesień 2023

Streszczenie

Praca rozważa problematykę skanerów bezpieczeństwa aplikacji internetowych. Wymienione zostają ich cechy wspólne oraz rozbieżne. Efektem pracy jest koncepcja i implementacja Generycznego skanera bezpieczeństwa, który pozwala na wykorzystanie istniejących, zewnętrznych narzędzi penetracyjnych do szerokiego testowania aplikacji pod kątem podatności.

Słowa kluczowe: cyberbezpieczeństwo, skanery bezpieczeństwa, testy penetracyjne

Abstract

This thesis covers topic of internet application security scanners. The paper considers their common and divergent characteristics. The result of the work is the concept and implementation of the Generic security scanner solution, which allows the use of existing, external penetration testing tools for comprehensive security assessments of internet applications.

Keywords: cybersecurity, security scanners, penetration testing

Spis treści

1. Wstęp

Obecnie na świecie cyfryzacja staje się coraz powszechniejsza. Jest to związane między innymi z powszechniejącym dostępem do internetu, do którego, według szacunków, w styczniu 2023 już 5,16 miliarda osób miało dostęp [1]. Tym samym na świecie dostępnych staje się coraz więcej usług, a wiele spraw, zamiast fizycznie, można załatwić przez internet [2]. W obliczu rozwoju użycia technologii internetowych i prawdopodobieństwa utrzymania powyższych trendów, coraz ważniejszym aspektem staje się cyberbezpieczeństwo.

Wraz ze wzrostem popularności m.in. cyfrowych usług publicznych i zakupów internetowych, coraz więcej danych użytkowników jest przechowywanych na serwerach powiązanych z aplikacjami webowymi. Informacje, takie jak dane teleadresowe czy dane o kartach płatniczych, są częstym celem hakerów, którzy chcą osiągnąć korzyści majątkowe [3]. Również powszechność usług finansowych dostępnych z poziomu komputera powiększa skalę ryzyka związanego ze złamaniem zabezpieczeń któregoś z systemu informatycznego. Nie można też zapominać o prywatności użytkowników, którzy korzystając z serwisów dostępnych globalnie, przesyłają między sobą ogromną ilość informacji. Mowa tu o komunikatorach, na których znajdują się np. wiadomości i zdjęcia, ale także o dyskach *w chmurach*, które magazynują ogromne ilości danych - również tych poufnych. Ich wyciek, w zależności od rodzaju danych, może mieć minimalne bądź też niezwykle poważne konsekwencje. Ludzie z obszaru cyberbezpieczeństwa próbują zapobiegać takim sytuacjom, a tym samym muszą nadążać za rozwojem cyfryzacji i zapewniać bezpieczeństwo *wirtualnemu światu*.

1.1 Cel

Celem tej pracy jest omówienie jednego z zagadnień związanego z cyberbezpieczeństwem, jakim są skanery podatności. Omówiono ich zastosowanie, różnice między ich poszczególnymi typami oraz ich wady i zalety. Efektem pracy jest koncepcja oraz implementacja generycznego skanera, którego działanie opiera się o wykorzystanie istniejących już narzędzi. Przeanalizowana została problematyka dotycząca takiego skanera oraz zaproponowane zostały rozwiązania poszczególnych problemów.

1.2 Organizacja pracy

Wstęp pracy skupia się na wprowadzeniu do szeroko pojętego cyberbezpieczeństwa. Zostały wskazane informacje dotyczące spektrum podatności, oraz jakie najczęściej znajdowane są w aplikacjach internetowych. Opisane są zagrożenia niesione przez dane podatności oraz przeanalizowane są statystyki o atakach na popularne aplikacje webowe. Rozdział porusza aspekt powtarzalności podatności i pokazuje, że często nie są one jednostkowymi przypadkami. Wyliczone są także ogólnodostępne źródła informacji na temat zidentyfikowanych zagrożeń.

Rozdział drugi omawia tematykę skanerów bezpieczeństwa. Wskazano przykładowe rozwiązania otwarte oraz komercyjne. Pokazano także różnice pomiędzy różnymi skanerami i ich odmienne zastosowania. Dla każdego rozwiązania wyliczono jego kluczowe cechy, zalety oraz wady.

Rozdział trzeci przedstawia koncepcję generycznego skanera bezpieczeństwa, który ma rozwiązać niektóre problemy ogółu narzędzi tego typu. Opisana jest motywacja, którą kierowano się w toku tworzenia niniejszej pracy. Pokazane są główne założenia i cele do spełnienia przez opisaną koncepcję. Sam skaner jest także przedstawiony od strony dostępnych funkcjonalności oraz przykładów użycia. Skupiono się także na opisaniu mechanizmu

modularności (wtyczki) oraz problemach, które rozwiązuje. Znajdują się tutaj również informacje na temat mobilności rozwiązania poprzez użycie konteneryzacji. Większość rozdziału zawiera opis rozwiązań na problemy zidentyfikowane w toku tworzenia koncepcji.

Rozdział czwarty omawia technologie wykorzystane w pracy. Są to m.in. użyty język programowania, systemy konteneryzacji, zewnętrzne aplikacje oraz oprogramowanie wspierające zarówno w procesie projektowania architektury/implementacji skanera, jak i w procesie powstawania niniejszej pracy pisemnej.

Rozdział piąty opisuje techniczne aspekty implementacji Generycznego skanera bezpieczeństwa, powstałej w trakcie tworzenia pracy. Zawarta została szczegółowa architektura rozwiązania, która jest opisana wraz z wyjaśnieniami co do podjętych decyzji projektowych. Pokazane są diagramy, które pomagają zrozumieć budowę rozwiązania. Została omówiona implementacja architektury związanej z wtyczkami i ogółem możliwości rozwoju aplikacji pod ich kątem.

Ostatni rozdział został poświęcony podsumowaniu pracy i omówieniu problemów, które zostały lub nie zostały rozwiązane. Nakreślono plany na dalszy rozwój aplikacji w różnych aspektach. Przedstawione są najważniejsze zalety oraz wady stworzonego rozwiązania. Pod tym kątem omówiono, które z nich wynikają z architektury, a które z samej implementacji. Sformułowano ogólne wnioski obejmujące całą pracę.

1.3 Podatności

Podatność (w kontekście cyberbezpieczeństwa) to słabość, luka lub błąd w oprogramowaniu, systemie, sieci lub procedurach, który może być wykorzystany przez cyberprzestępców, aby uzyskać nieautoryzowany dostęp, wprowadzić zmiany, wykraść dane lub zakłócić działanie systemu. Podatności stanowią ryzyko dla bezpieczeństwa informacji, ponieważ mogą prowadzić do naruszenia integralności, poufności lub dostępności danych oraz usług (*CIA triad: Confidentiality, Integrity, Availability*).

Podatności mogą występować z różnych przyczyn, np.:

- **Błędy w sztuce programowania**: Nieumyślne pomyłki w kodzie źródłowym oprogramowania, które mogą prowadzić do nieprawidłowego działania programu, umożliwiając atakującemu wykorzystanie błędu.
- **Zaniedbania konfiguracyjne**: Nieprawidłowa konfiguracja systemów, urządzeń lub sieci może prowadzić do podatności, takich jak ujawnienie wrażliwych informacji czy pozostawione otwarte porty, które mogą być wykorzystane przez atakującego.
- **Braki w procesach i procedurach**: Niewłaściwe praktyki zarządzania bezpieczeństwem, takie jak słabe hasła, brak aktualizacji lub brak regularnych przeglądów bezpieczeństwa.
- **Podatności w zewnętrznych komponentach**: Korzystanie z nieaktualnego lub podatnego oprogramowania - takiego jak biblioteki, wtyczki czy moduły - może prowadzić do podatności, które wpływają na cały system.

  Pojęcie to jest wykorzystywane na przestrzeni całej pracy.

1.4 Klasyfikacja podatności ze względu na typ

Choć liczba znalezionych podatności, z uwagi na powiększającą się liczbę aplikacji, stale rośnie, to można je uporządkować korzystając z istniejących metod klasyfikacji. Należy jednak mieć na uwadze to, że pomaga ona jedynie w sprecyzowaniu charakteru występujących

słabości. Przykładowo, widząc podatność typu Cross-site Scripting (XSS) można być pewnym, że w podatnej aplikacji możliwe jest wykonanie kodu po stronie przeglądarki klienta, co może być użyte do innych, bardziej specyficznych dla aplikacji ataków [4]. Z kolei podatności związane z kontrolą dostępu zazwyczaj prowadzą do innych efektów, takich jak ujawnienie wrażliwych danych czy wykonanie nieuprawnionej akcji [5]. Istnieją różne organizacje, które opracowują własne listy typów podatności.

1.4.1 OWASP

OWASP (*Open Web Application Security Project*) to organizacja non-profit, która zajmuje się badaniem i popularyzowaniem wiedzy na temat bezpieczeństwa aplikacji internetowych. Na jej stronie internetowej można znaleźć opisy różnych podatności, wraz z informacjami, jak je usunąć lub im zapobiegać.

OWASP regularnie tworzy listę OWASP Top 10[6]. To zbiór podatności, które są najczęściej wykorzystywane przez hakerów i stanowią największe zagrożenie dla bezpieczeństwa aplikacji internetowych. *OWASP Top 10* jest narzędziem nie tylko dla osób zajmujących się cyberbezpieczeństwem, ale także dla programistów, projektantów, architektów i menedżerów, którzy chcą zapewnić bezpieczeństwo swoich aplikacji internetowych i zminimalizować ryzyko naruszeń bezpieczeństwa. Publikowanie takiej listy może pomóc w ustaleniu *trendów*, jeśli chodzi o ataki na aplikacje webowe, przez co organizacje mogą skupić się na potencjalnych problemach, które stanowią największe ryzyko. Najnowsza wersja listy *OWASP Top 10* pochodzi z 2021 roku i znajduje się poniżej:

- A01:2021 - Broken Access Control,
- A02:2021 - Cryptographic Failures,
- A03:2021 - Injection,
- A04:2021 - Insecure Design,
- A05:2021 - Security Misconfiguration,
- A06:2021 - Vulnerable and Outdated Components,
- A07:2021 - Identification and Authentication Failures,
- A08:2021 - Software and Data Integrity Failures,
- A09:2021 - Security Logging and Monitoring Failures,
- A10:2021 - Server-Side Request Forgery (SSRF).

1.4.2 CWE

CWE (*Common Weakness Enumeration*)[7] to standardowa lista opisów podatności w oprogramowaniu, która jest rozwijana przez organizację MITRE Corporation we współpracy z ekspertami z całego świata. *CWE* zawiera opisy ponad 1 000 podatności i błędów w oprogramowaniu, a każda z tych podatności jest przypisana do unikalnego identyfikatora numeru *CWE*. Lista umożliwia opisanie podatności w oprogramowaniu na różnych poziomach abstrakcji, od bardziej ogólnych opisów po bardziej szczegółowe. W wyniku tego, jedna podatność może rozbijać się na wiele bardziej specyficznych podatności, co umożliwia dokładniejsze określenie zagrożenia. W jednym numerze *CWE* mogą zawierać się inne numery *CWE*, w przypadku gdy opisana podatność jest złożona lub składa się z kilku elementów, które można opisać jako oddzielne podatności. W takim przypadku główny numer *CWE* odnosi się do całej podatności, podczas gdy numery podrzędne odnoszą się do konkretnych aspektów tej podatności.

Przykładowo, *CWE-601* (*URL Redirection to Untrusted Site ('Open Redirect')*) jest podatnością, która może obejmować wiele różnych sposobów wykorzystania, w tym atak typu phishing lub przekierowanie użytkownika na niebezpieczną stronę internetową. W tym przypadku *CWE-601* odnosi się do całej podatności, a numery podrzędne (*children*) odnoszą się do konkretnych aspektów podatności, takich jak konkretny sposób wykorzystania przekierowania URL [8].

Inny przykład to *CWE-89* (*SQL Injection*), która odnosi się do kategorii podatności, ale zawiera wiele podatności podrzędnych. Sam numer *CWE-89* obejmuje różne aspekty związane z wstrzykiwaniem kodu SQL, a numery podrzędne odnoszą się do konkretnych przypadków wykorzystania tej podatności w różnych językach programowania, frameworkach i bibliotekach. Z drugiej strony, *CWE-89* jest *dzieckiem CWE-943* (*Improper Neutralization of Special Elements in Data Query Logic*). Taki sposób klasyfikacji pozwala na dokładniejsze opisanie skomplikowanych podatności, uwzględniając ich różnorodność i specyfikę[9].

Podobnie jak w przypadku *OWASP*, istnieje lista najczęściej występujących *CWE* pod nazwą *CWE Top 25 Most Dangerous Software Weaknesses*[10].

1.5 CVE

CVE (*Common Vulnerabilities and Exposures*)[11] to system, który umożliwia identyfikację i klasyfikację znanych luk bezpieczeństwa w oprogramowaniu. Jest zarządzany przez organizację *MITRE*, która współpracuje z publicznymi i prywatnymi instytucjami, aby gromadzić i udostępniać informacje na temat owych luk. Każde odkrycie jest oznaczone unikalnym identyfikatorem *CVE*, który składa się z:

- liter `CVE`,
- roku, w którym luka została zgłoszona,
- unikalnego numeru przypisanego do tej konkretnej luki.

Każda z powyższych jest oddzielona znakiem myślnika. Tak wygląda przykładowy numer *CVE*: `CVE-2023-12345`.

CVE między innymi pomaga organizacjom w weryfikacji informacji o bezpieczeństwie używanego oprogramowania. Identyfikatory *CVE* są powszechnie używane przez profesjonalistów w dziedzinie bezpieczeństwa do szybkiego wyszukiwania informacji o konkretnych podatnościach i śledzenia ich napraw. *CVE* jest szczególnie użyteczne w koordynowaniu wdrażania poprawek, ponieważ odnosi się do jednej, konkretnej luki.

1.6 Security advisories

Zalecenia dotyczące bezpieczeństwa (*Security advisories*) to oficjalne ogłoszenia wydawane przez producentów oprogramowania, organizacje bezpieczeństwa lub dostawców usług. Ich celem jest informowanie użytkowników, administratorów systemów i profesjonalistów zajmujących się bezpieczeństwem o znalezionych lukach bezpieczeństwa, podatnościach oraz potencjalnych zagrożeniach związanych z oprogramowaniem lub systemami. Zalecenia dotyczące bezpieczeństwa często zawierają informacje, takie jak:

- **Opis podatności**: Krótki opis luki bezpieczeństwa, wraz z identyfikatorem *CVE* (jeśli istnieje),
- **Wpływ na bezpieczeństwo**: Ocena ryzyka związanego z podatnością, zwykle zgodna z systemem oceny ryzyka takim jak *CVSS* (*Common Vulnerability Scoring System*),

- **Dotknięte systemy**: Lista systemów operacyjnych, aplikacji lub konfiguracji, które są narażone na daną podatność,
- **Rozwiązania i obejścia**: Instrukcje dotyczące naprawy luki lub zastosowania tymczasowych rozwiązań mających na celu zmniejszenie ryzyka, zanim zostanie wydana oficjalna łatka,
- **Aktualizacje**: Informacje o dostępnych aktualizacjach, które naprawiają podatność.

Security advisories zazwyczaj są bardziej precyzyjne niż informacje zawarte w *CVE*, ponieważ zazwyczaj pisze je producent oprogramowania. Przykład zaleceń dotyczących bezpieczeństwa można znaleźć na stronie popularnego CMS *Drupal* [12].

1.7 Przykład WordPressa

Na podstawie statystyk opublikowanych przez W3Techs wynika, że 39.1% wszystkich stron internetowych, to strony stworzone na WordPressie. W samym rynku CMS udział *WordPress* wynosi 63.8%, co czyni go liderem. Platformy będące dla niego konkurencją osiągają wyniki jednocyfrowe, zarówno jeśli chodzi o odsetek wszystkich stron internetowych, jak i procentowy udział w rynku [13].

Wysoka popularność w przypadku *WordPressa* wiąże się także z wysoką ilością infekcji stron internetowych, działających pod jego kontrolą. Wiele statystyk dotyczących bezpieczeństwa tego CMS publikuje firma *Sucuri*, która zajmuje się zabezpieczaniem stron internetowych oraz jest autorem wtyczki *Sucuri Security*, który posiada ponad 800 000 aktywnych instalacji [14]. Statystyki te pokazują, że w 2019 roku 94% infekcji stron opartych o systemy CMS dotyczyło tych działających na WordPressie [15]. Wyraźnie większa ilość infekcji dotyczących tej platformy wynika między innymi z dużych podobieństw między realnymi instalacjami stron internetowych. W przypadku odnalezienia podatności w silniku *WordPress*, podatna może zostać każda witryna. Podobnie dzieje się z wtyczkami, które mają dodawać dodatkowe funkcje do strony internetowej - gdy podatność zostanie odnaleziona w jednej z popularniejszych, to ryzyko może dotyczyć dużej grupy stron internetowych. W przypadku, gdy prawie połowa stron internetowych działa na WordPressie [13], rośnie możliwość wykorzystania narzędzi automatyzujących ataki na wybraną podatność, która mogła jeszcze nie zostać załatana (tzw. 0-day) lub gdy aktualizacja nie została jeszcze wgrana przez właściciela strony.

Raport *Sucuri* zawiera także statystyki dotyczące odsetka zainfekowanych stron internetowych, które nie były zaktualizowane do najnowszej wersji. Z ogólnych statystyk wynika, że 44% zainfekowanych stron było aktualnych, natomiast 56% nie. Są także statystyki, które dodatkowo uwzględniają system CMS. Na 11 porównanych, *WordPress* zajął ostatnie miejsce z najmniejszą liczbą infekcji na nieaktualnych wersjach i wynikiem 49%. Dla przykładu, CMSy takie jak *Drupal* czy *Joomla* osiągnęły wyniki kolejno 77% i 90%[15]. Różnice te wynikają z faktu, że *WordPress* wraz z wersją 3.7 wprowadził automatyczne aktualizacje swojego rdzenia (*core*) [16]. Oznacza to, że witryny z *WordPressem* w wersjach wydanych po 24.10.2013 domyślnie aktualizowały się samodzielnie.

Powyższe dane pokazują, że znalezienie podatności w jednym frameworku może stanowić istotne zagrożenie dla wielu aplikacji działających w internecie. Opisany przykład podsumowuje potrzebę nadawania unikalnych numerów podatności. Właściciele serwisów, wiedząc o istnieniu podatności pod danym numerem *CVE*, mogą zweryfikować (np. na stronie producenta), która wersja oprogramowania usunęła tę konkretną lukę. Jednocześnie skanery bezpieczeństwa mogą przeprowadzać automatyczną weryfikację i raportować operatorom wyniki dotyczące testów na konkretne podatności. Operator skanera bowiem musi wiedzieć,

czy podatność w ogóle była testowana - sam wynik skanera, że strona jest bezpieczna nie jest wystarczający.

2. Skanery podatności

Skanery bezpieczeństwa to narzędzia informatyczne służące do identyfikowania, analizowania i raportowania potencjalnych zagrożeń oraz podatności w sieciach komputerowych, systemach i aplikacjach. Skanery te pozwalają specjalistom ds. bezpieczeństwa monitorować i utrzymywać zabezpieczenia w organizacji, pomagając w ochronie przed atakami hakerów, oprogramowaniem szpiegującym, malware i innymi zagrożeniami.

2.1 Zastosowanie skanerów bezpieczeństwa

Ze względu na regulacje krajowe oraz globalne (np. Rozporządzenie o Ochronie Danych Osobowych, RODO [17]), firmy muszą stosować się do rygorystycznych wymogów dotyczących bezpieczeństwa danych oraz regularnie udowadniać zgodność z tymi regulacjami. Sama ochrona danych osobowych polega w dużej mierze na monitorowaniu bezpieczeństwa systemów informatycznych, w których te informacje są przechowywane, bądź przez które są przetwarzane. Choć korzystanie ze skanerów bezpieczeństwa nie jest rozwiązaniem wszystkich problemów, może ono pozytywnie wpłynąć na stan bezpieczeństwa systemów informatycznych, poprzez wczesne wykrywanie podatności i informowanie o nich odpowiedzialnych osób. Skanery bezpieczeństwa pozwalają na automatyzację niektórych manualnych procesów, takich jak weryfikacja aktualności użytych komponentów. Bardziej zaawansowane skanery mają na celu wsparcie testów penetracyjnych poprzez wyszukiwanie konkretnych podatności. Skanerów nie można jednak traktować tak samo, z powodu różnic w skuteczności wykrywania przez nich podatności [18]. Skanery podatności bazują na listach opublikowanych w formie numerów CVE (*Common Vulnerabilities and Exposures*) lub próbują odnaleźć podatności w aplikacji samodzielnie.

2.2 Podział skanerów ze względu na obszar zastosowania

Skanery bezpieczeństwa można podzielić na kilka kategorii, w zależności od obszaru zastosowania i celów, dla których są używane [19]. Przykładowy podział:

- **Skanery sieciowe**: skanery analizujące bezpieczeństwo sieci komputerowych, skupiając się na podatnościach związanych z protokołami, usługami i konfiguracją urządzeń sieciowych,
- **Skanery systemów operacyjnych**: skanery oceniające bezpieczeństwo konkretnych systemów operacyjnych, identyfikując podatności związane z ich aktualnością oraz konfiguracją,
- **Skanery kodu źródłowego**: skanery analizujące kod źródłowy aplikacji m.in. pod kątem podatności związanych z błędami programistów, użycia podatnych komponentów lub niepoprawnej konfiguracji,
- **Skanery aplikacji internetowych**: skanery weryfikujące bezpieczeństwo aplikacji dostępnych z poziomu internetu.

Przedstawiony podział nie może być uznany za precyzyjny choćby dlatego, że niektóre skanery przenikają się funkcjami oraz zastosowaniem - np. skanery aplikacji internetowych często posiadają funkcje związane ze skanowaniem sieci, które wykonywane jest w celu odnalezienia aplikacji internetowych. Skanery mogłyby też zostać podzielone ze względu na platformy - w istocie skanery kodu źródłowego będą się znacznie różniły pomiędzy aplikacjami webowymi, a tymi, które są pisane na urządzenia mobilne. Z kolei skanery

aplikacji na urządzenia mobilne mogłyby zostać ponownie podzielone na takie, które zajmują się konkretnymi systemami operacyjnymi, ponieważ każdy z nich posiada inną architekturę i własne podejście do kwestii bezpieczeństwa [20].

Mimo to przedstawiony podział pozwala na sprecyzowanie rodzajów skanerów, które obejmuje niniejsza praca: są to skanery sieciowe oraz skanery do aplikacji internetowych.

2.3 Podział skanerów bezpieczeństwa aplikacji internetowych na typy

Skanery bezpieczeństwa aplikacji internetowych można podzielić na kilka głównych typów, w zależności od tego, jak i kiedy są stosowane. Oto najpopularniejsze:

- **Skanery *DAST* (*Dynamic Application Security Testing*)**: Te skanery analizują aplikację podczas jej działania, testując ją na obecność różnych luk bezpieczeństwa. Są one skuteczne w wykrywaniu problemów, które mogą wystąpić podczas interakcji użytkownika z aplikacją, takich jak *Cross-Site Scripting (XSS)* czy *SQL Injection*[21].
- **Skanery SAST *(Static Application Security Testing*)**: Skanery te analizują kod źródłowy aplikacji w poszukiwaniu potencjalnych luk bezpieczeństwa. Mogą one wykryć problemy, które mogą nie być widoczne podczas działania aplikacji, ale mogą stanowić zagrożenie, gdy kod jest wykonany [22].
- **Skanery IAST (*Interactive Application Security Testing*)**: Te skanery łączą aspekty *DAST* i *SAST*, analizując kod źródłowy podczas działania aplikacji. Dzięki temu mogą one wykryć więcej problemów niż każde z tych narzędzi osobno [23].
- **Skanery RASP (*Runtime Application Self-Protection*)**: RASP to technologia, która działa w środowisku uruchomieniowym aplikacji, monitorując jej działanie i blokując podejrzane aktywności w czasie rzeczywistym [24].
- **Skanery SCA (*Software Composition Analysis*)**: Te skanery analizują komponenty oprogramowania używane przez aplikację (np. biblioteki, frameworki) w poszukiwaniu znanych luk bezpieczeństwa [25].
- **Narzędzia penetracyjne (*Penetration Testing Tools*)**: Te narzędzia są używane do przeprowadzania symulowanych ataków na aplikację w celu identyfikacji potencjalnych luk bezpieczeństwa.

Niniejsza praca obejmuje skanery *DAST* oraz narzędzia penetracyjne.

2.4 Skanery DAST

DAST, czyli *Dynamic Application Security Testing*, to technologia skanowania, która testuje aplikacje internetowe w czasie ich działania (*runtime*) w celu znalezienia potencjalnych luk bezpieczeństwa. *DAST* jest zazwyczaj stosowany do wykrywania luk w zabezpieczeniach na etapie produkcji. Oto kilka kluczowych cech i aspektów skanerów DAST [26]:

- **Testowanie w czasie rzeczywistym**: Skanery *DAST* testują aplikacje podczas ich działania, co pozwala na wykrywanie luk, które mogą wystąpić podczas interakcji użytkownika z aplikacją.
- **Wykrywanie typowych luk bezpieczeństwa**: Skanery *DAST* są skuteczne w wykrywaniu typowych luk bezpieczeństwa aplikacji internetowych, takich jak *Cross-Site Scripting (XSS)*, ataki typu "wstrzyknięcie" (np. *SQL Injection*), i inne.
- **Testowanie w formie czarnej skrzynki**: *DAST* jest często nazywany "czarnoskrzynkowym" (*black-box*) podejściem do testowania bezpieczeństwa, ponieważ

nie wymaga dostępu do kodu źródłowego aplikacji. Zamiast tego, skanery *DAST* testują aplikacje z perspektywy użytkownika końcowego.

- **Automatyzacja**: Skanery *DAST* często oferują automatyzację, co pozwala na regularne skanowanie aplikacji w celu ciągłego monitorowania bezpieczeństwa.
- **Raportowanie i łatwość użycia**: Większość skanerów *DAST* oferuje generowanie raportów, które mogą pomóc zespołom deweloperskim szybko zidentyfikować i naprawić znalezione luki bezpieczeństwa.

Przykładowy schemat działania skanerów *DAST*:

1. **Skanowanie struktury aplikacji**: Skanery rozpoczynają od analizy struktury aplikacji webowej, w tym wszystkich dostępnych stron, formularzy, plików konfiguracyjnych i innych elementów. Mogą korzystać z różnych technik, takich jak przeszukiwanie rekurencyjne, analiza mapy witryny (*sitemap*) lub analiza kodu źródłowego.
2. **Analiza podatności**: Skanery wykorzystują różne techniki i metody w celu identyfikacji podatności w aplikacji. Mogą przeprowadzać automatyczne testy penetracyjne, analizować kod źródłowy, sprawdzać poprawność konfiguracji serwera, weryfikować poprawność uwierzytelniania i autoryzacji, analizować podatności związane z wstrzykiwaniem kodu (np. *SQL Injection*) i wiele innych.
3. **Wykrywanie podatności**: Na podstawie analizy, skaner generuje raport zawierający informacje o znalezionych podatnościach. Raport może obejmować opis podatności, jej lokalizację, kroki reprodukcji, dowody potwierdzające wystąpienie i zalecenia dotyczące naprawy.
4. **Testowanie wydajnościowe**: Niektóre skanery mogą również przeprowadzać testy wydajnościowe, testy obciążeniowe i testy skali, w celu sprawdzenia, jak aplikacja reaguje na różne obciążenia. Testowanie wydajnościowe może być przydatne do identyfikacji podatności związanych z odrzucaniem usług (*Denial of Service*) i innych zagrożeń związanych z wydajnością aplikacji.

Nie każdy skaner wykona wszystkie z podanych kroków, ponieważ zależy to od zamysłu jego twórcy i dostępnych funkcjonalności.

Przykłady skanerów *DAST*: *Nikto* [27], *Nuclei* [28], *OpenVAS* [29].

2.5 Kompleksowe skanery bezpieczeństwa

Poruszając tematykę skanerów *DAST* należy zauważyć, że samo skanowanie aplikacji to często nie jest jedyna funkcjonalność dostępnych na rynku narzędzi. Na skaner *DAST* można zatem patrzeć jako część większego skanera.

Na potrzebę tej pracy zostanie wprowadzone pojęcie kompleksowego skanera bezpieczeństwa. To skaner, który wykonuje wiele różnych zadań w sposób równoległy lub sekwencyjny. Jest on nastawiony na wykonanie jak największej liczby czynności bez udziału operatora. Typowym scenariuszem użycia będzie podanie adresu IP lub nazwy hosta do skanowania i uruchomienie skanera. Skaner powinien automatycznie wykryć uruchomione na serwerze serwisy i przeskanować je pod kątem podatności, nie ograniczając się wyłącznie do aplikacji webowych, choć także je uwzględniając. Z tego powodu kompleksowy skaner bezpieczeństwa będzie posiadał wbudowany skaner typu DAST.

2.5.1 Zalety kompleksowych skanerów bezpieczeństwa

- **Łatwość obsługi**: Obsługa takiego skanera w większości sytuacji nie wymaga dużej wiedzy technicznej. Domyślna konfiguracja powinna wystarczyć dla typowego zastosowania.
- **Brak potrzeby znajomości działających usług**: Operator nie musi wiedzieć, co działa na danym hoście, aby użyć takiego skanera. Uruchomienie takiego skanera nie wymaga zatem wcześniejszego rekonesansu ze strony operatora, które zajęłoby czas.
- **Kompleksowość działania**: Narzędzia tego typu zawierają dużo funkcjonalności i kontrolek (tzw. *checków*). Dzięki temu w prosty sposób można wykonać wiele czynności na raz, co jest kolejnym sposobem na zaoszczędzenie czasu.
- **Prosta instalacja**: Porównując takie rozwiązanie do kilku oddzielnych narzędzi, które mogłyby wykonać podobne zadania, należy zwrócić uwagę na fakt, że instalacja jednego programu będzie prostsza niż kilku. Utrzymanie jednego narzędzia (np. poprzez aktualizacje) powinno być również prostsze niż wielu.
- **Magazynowanie danych i ich ponowne używanie**: Kompleksowe narzędzia często zbierają dużo informacji i wykorzystują je na potrzeby późniejszych procedur. Przykładowo, odnalezienie działających serwisów może w przyszłości przyśpieszyć wykonywanie ponownych skanów. Dane zebrane przez dany krok skanera (*output*) mogą stanowić dane wejściowe dla kolejnego (*input*). W przypadku użycia takiego narzędzia, operator nie musi dokonywać procedury przekształcania zebranych danych z formatu wyjściowego do formatu wejściowego, ponieważ odpowiednio napisane kompleksowe narzędzie będzie to robić we własnym zakresie.

2.5.2 Wady kompleksowych skanerów

- **Niska znajomość wewnętrznych działań**: Kompleksowe skanery są często rozwiązaniami zamkniętymi, zatem używa się ich jako czarnych skrzynek (*black-boxes*) - uruchamia się i korzysta bez świadomości, co, gdzie i w jaki sposób jest wykonywane. Nie wiedząc jak dokładnie wyglądają testy, ciężko odnieść się do efektywności takiego skanowania, zatem użytkownicy mocno polegają na reputacji producenta.
- **Iluzja kompleksowości**: Użycie skanera, który ma skanować wszystko, daje poczucie bezpieczeństwa. Dodatkowo, marketing twórców tego typu oprogramowania może tworzyć iluzoryczne wrażenie, że jest to całkowite rozwiązanie problemów związanych z bezpieczeństwem.
- **Brak otwartości**: Wiele kompleksowych skanerów to rozwiązania komercyjne, których nie można modyfikować np. dopisując własne kontrolki (*checks*).
- **Duże wykorzystanie zasobów**: Skanowanie w znacznym stopniu jest procesem iteracyjnym, a skanowanie aplikacji pod kątem dużej ilości podatności może konsumować wiele zasobów sieciowych, zasobów komputera oraz czasu. Kompleksowe skanery uruchamiając skanowanie mogą weryfikować system pod kątem archaicznych podatności, które z jednej strony mogą nieść ogromne ryzyko, ale z drugiej ich wystąpienie jest w praktyce niemożliwe.
- **Cena**: W związku z tym, że wiele kompleksowych skanerów to rozwiązania komercyjne, należy zwrócić uwagę na ich cenę, która może stanowić zbyt wysoką barierę wejścia dla niektórych organizacji.

Przykłady kompleksowych skanerów: *Nessus* [30], *Qualys* [31] (wiele narzędzi), *Acunetix* [32], *OpenVAS* [29].

2.6 Narzędzia penetracyjne

Narzędzia penetracyjne (*Penetration Testing Tools, Pentest Tools*) są narzędziami stosowanymi w cyberbezpieczeństwie do testowania i identyfikacji luk w systemach bezpieczeństwa. Stosowane są przez pentesterów podczas testów penetracyjnych.

Testy penetracyjne (*Penetration Tests, Pentests*) są częścią skoordynowanych ataków na systemy informatyczne, przeprowadzanych przez zespół pentesterów, który ma za zadanie znaleźć słabe punkty aplikacji, zanim zrobią to osoby o złych zamiarach. Wynikiem testów jest szczegółowy raport z listą znalezionych podatności, który można przekazać osobom odpowiedzialnym za utrzymanie i rozwój testowanej aplikacji [33].

Narzędzia penetracyjne obejmują różne rodzaje oprogramowania i sprzętu. Przykłady to skanery sieciowe, które identyfikują otwarte porty i lukratywne cele, narzędzia do łamania haseł próbujące złamać uwierzytelnianie, oraz narzędzia do analizy kodu, które szukają luk w oprogramowaniu. Cechy charakterystyczne narzędzi penetracyjnych:

- Z uwagi na to, że narzędzia te są wykorzystywane przez specjalistów, często charakteryzują się licznymi opcjami konfiguracji.
- Samo używanie narzędzia może stanowić wyzwanie dla mniej doświadczonych osób (np. ze względu na interfejs oparty jedynie na wierszu poleceń (*Command Line Interface, CLI*).
- Twórcami narzędzi często są pojedyncze osoby lub inni pentesterzy, którzy dla ułatwienia pracy stworzyli narzędzie, które następnie opublikowali.
- Niektóre narzędzia koncentrują się na tylko jednej podatności. Takie narzędzie może być opublikowane w formie dowodu na możliwość wykorzystania jakiejś podatności (*Proof of Concept, PoC)*).
- Może być wymagana znajomość testowanej aplikacji.
- Większość narzędzi jest otwartoźródłowa (*open-source*), zatem możliwe jest dostosowanie narzędzia pod konkretny test.
- Mogą być bardziej inwazyjne niż skanery. W przeciwieństwie do nich, mogą próbować wykorzystać podatność, a nie tylko zasygnalizować jej potencjalne istnienie.

Narzędzie penetracyjne może być również rodzajem skanera. Niektóre skanery (np. skaner wbudowany w Burp Suite) można nazwać narzędziem penetracyjnym, ponieważ całe oprogramowanie *Burp Suite* jest narzędziem przeznaczonym do testów bezpieczeństwa [34]. Innym przykładem jest *wpscan*, który skanuje instancje CMS *WordPress* pod kątem różnych podatności [35].

Bardzo popularnym narzedziem pentesterskim jest *Metasploit Framework*, który poprzez wbudowane lub zewnętrzne moduły jest w stanie wykorzystać wiele podatności w danej aplikacji [36]. Narzędzie nie może być jednak uznane za skaner, ponieważ wymaga ciągłej interakcji - każdy moduł uruchamiany jest oddzielnie i wymaga różnych opcji konfiguracyjnych. Z drugiej strony, narzędzie tworzy swoją bazę danych o hoście w trakcie korzystania z modułów. Przykładowo, jeśli modułowi wykonującemu atak siłowy uda się zalogować na konto, to *Metasploit Framework* zapisze te dane w swojej bazie i będzie można z nich korzystać później (również w innych modułach)[36].

2.7 Opisy konkretnych rozwiązań

Jak wskazano powyżej, skategoryzowanie skanerów nie zawsze jest proste, i wiele skanerów może być jednocześnie narzędziem penetracyjnym i skanerem DAST. Z tego powodu w tej sekcji zostaną opisane wszystkie rodzaje skanerów.

Narzędzia użyto na *Damn Vulnerable WordPress (DVWP)* - celowo podatnej instancji *WordPress* [37]. Zawiera ona liczne podatności i błędy konfiguracyjne. W zależności od wygody uruchomienia narzędzia penetracyjnego bądź skanera, instancja *DVWP* działała w lokalnym kontenerze *Docker* lub na zewnętrznym serwerze wirtualnym (*VPS*).

Należy podkreślić, że niniejsza praca nie obejmuje porównywania dokładnej skuteczności skanerów, a zawarte w tym punkcie rezultaty mają stanowić punkt odniesienia z perspektywy ogólnych metod podejścia do problemu wyszukiwania podatności oraz używania narzędzia.

2.7.1 Burp Suite Pro

Pakiet narzędzi *Burp Suite* jest bardzo rozpoznawalnym narzędziem w świecie cyberbezpieczeństwa, a w szczególności tym związanym z testowaniem aplikacji webowych [34]. Fundamentem tego programu jest *Proxy*. Z definicji, *proxy* to urządzenie lub program zapewniający komunikację pomiędzy klientem a serwerem [38]. W przypadku aplikacji webowych stosuje się *web-proxy*, które pośredniczy w przesyle danych pomiędzy przeglądarką użytkownika a serwerem. Pozwala to m.in. na monitorowanie ruchu poprzez wgląd w poszczególne zapytania oraz ich modyfikację *w locie*. Proxy jest w stanie zatrzymać wysłane zapytanie, aby użytkownik *web-proxy*, takiego jak *Burp Suite*, mógł je np. zmodyfikować i przesłać dalej [39].

Burp Suite w wersji Pro posiada wbudowany *DAST*, który może zostać skonfigurowany na wiele sposobów, z czego najistotniejsze metody konfiguracji to:

- ***Live Passive Crawl from Proxy***: Polega na pasywnym przechwytywaniu i analizowaniu ruchu sieciowego przepływającego przez serwer *proxy Burp*. Skaner analizuje odpowiedzi serwera na żądania klienta, w celu wykrycia potencjalnych luk bezpieczeństwa, takich jak niewłaściwa obsługa danych wejściowych, niewłaściwa konfiguracja serwera lub ujawnianie informacji poufnych.
- ***Live Audit from Proxy***: Ta metoda skanowania jest podobna do *Live Passive Crawl from Proxy*, ale dodatkowo skaner wykonuje aktywne żądania do serwera w celu wykrycia potencjalnych luk bezpieczeństwa. Skaner może na przykład próbować manipulować danymi wejściowymi lub wykorzystywać znane luki bezpieczeństwa w oprogramowaniu serwera.
- ***Crawl and Audit***: Polega na pełnym przeszukaniu i audycie strony internetowej. Skaner najpierw pełza (*crawls*) po stronie internetowej, analizując strukturę strony i gromadząc informacje o dostępnych URL-ach, formularzach, ciasteczkach i innych elementach. Następnie skaner wykonuje audyt, wysyłając aktywne żądania do serwera w celu wykrycia potencjalnych luk bezpieczeństwa.
- ***Passive Scan***: W trakcie pasywnego skanowania, *Burp Suite Pro* analizuje ruch sieciowy, który przepływa przez serwer *proxy*, ale nie wysyła dodatkowych żądań do serwera. W ramach pasywnego skanowania *Burp Suite Pro* może wykryć potencjalne problemy z bezpieczeństwem, takie jak ujawnianie informacji poufnych, niewłaściwa obsługa danych wejściowych, czy niewłaściwa konfiguracja serwera. Pasywne skanowanie jest mniej inwazyjne niż aktywne skanowanie i nie wpływa na działanie aplikacji webowej, ale może nie wykryć wszystkich potencjalnych luk bezpieczeństwa.

- ***Active Scan***: Podczas aktywnego skanowania, *Burp Suite Pro* wysyła dodatkowe żądania do serwera, w celu wykrycia potencjalnych luk bezpieczeństwa. Aktywne skanowanie może wykryć szeroki zakres luk bezpieczeństwa, takich jak podatności na ataki *XSS*, *SQL Injection*, *CSRF*, czy *RCE*. Aktywne skanowanie jest bardziej inwazyjne niż pasywne skanowanie i może wpłynąć na działanie aplikacji webowej, ale jest również bardziej skuteczne w wykrywaniu luk bezpieczeństwa.

Pomimo tego, że *proxy* nie jest bezpośrednio powiązane z tematyką skanerów bezpieczeństwa, w przypadku *Burp Suite* istotne jest współdziałanie wielu narzędzi: dane zbierane przez *proxy* to najczęściej autentyczne akcje użytkownika wykonywane w aplikacji webowej. W momencie, w którym zapytania trafiają do *Burp Suite* zostają zapisane w historii oraz w drzewie strony (tzw. *sitemap*). Na podstawie zgromadzonych informacji, aplikacja jest w stanie wykonywać bardziej precyzyjne skany i dostosowywać się do konkretnych aplikacji.

Istnieje także bardziej zautomatyzowana wersja samego skanera, która jest dostępna w wersji *Burp Suite Enterprise*. W przeciwieństwie do desktopowych wersji *Community* oraz *Pro*, *Enterprise* jest aplikacją webową. W typowym scenariuszu uruchamia się ją na urządzeniu pełniącym funkcję serwera i uruchamia skany na wskazanych stronach internetowych. Skaner w tej wersji nie bierze pod uwagę działań manualnych użytkownika i po wstępnym skonfigurowaniu działa w pełni automatycznie. Aplikacja pozwala także na proste generowanie raportów ze skanów oraz ustalanie harmonogramu wykonywanych zadań. Tej wersji skanera użyto do przeprowadzenia przykładowego skanu na podatną instancję *WordPress*.

2.7.2 Nessus Professional

Nessus Professional to komercyjne narzędzie do skanowania podatności, które jest używane do zabezpieczania sieci przed znanymi i potencjalnymi lukami bezpieczeństwa. Twórcy skanera - *Tenable* - udostępniają także uboższą o niektóre funkcjonalności, ale darmową wersję skanera o nazwie *Nessus Essentials*[30], [40].

Oto niektóre główne cechy *Nessus Professional*:

- **Rozległa baza danych podatności**: *Nessus Professional* posiada jedną z największych baz danych podatności na świecie, co pozwala na identyfikację i klasyfikację wielu różnych rodzajów zagrożeń.
- **Wszechstronność**: *Nessus* może skanować szeroką gamę systemów, w tym komputery osobiste, serwery, urządzenia sieciowe i więcej. Może również skanować różne systemy operacyjne i sieci.
- **Raportowanie i analiza**: *Nessus Professional* oferuje szczegółowe raporty, które prezentują wyniki skanowania w łatwy do zrozumienia sposób. Może również integrować się z innymi narzędziami do zarządzania podatnościami i bezpieczeństwem.
- **Automatyzacja**: *Nessus Professional* oferuje możliwość automatyzacji skanowania podatności, co pozwala na regularne przeprowadzanie skanowań bez konieczności ręcznego uruchamiania procesu.

Nessus jest narzędziem, które potrafi skanować dany host pod kątem różnych serwisów. Dzięki temu, jak zostało to wspomniane wyżej, jest narzędziem wszechstronnym i nie ogranicza się do skanowania konkretnego typu serwisów (choć ich ilość jest również w pewnym stopniu ograniczona do tych popularnych). Narzędzie zatem może być wykorzystywane do weryfikacji poziomu bezpieczeństwa całej infrastruktury (np. serwerów mailowych, serwerów www, serwerów FTP, hostów użytkowników), a nie tylko aplikacji webowych. *Nessus*

posiada również wbudowany *DAST*, więc jest w stanie analizować aplikacje internetowe. Te funkcjonalności czynią skaner *Nessus Professional* kompleksowym skanerem.

2.7.3 Nuclei

Nuclei to narzędzie penetracyjne służące do zautomatyzowanego skanowania podatności, które pozwala na sprawdzanie bezpieczeństwa aplikacji i serwisów internetowych. *Nuclei* jest narzędziem o otwartym źródle kodu, co oznacza, że może być używane i modyfikowane za darmo [28]. Główne cechy *Nuclei* obejmują:

- **Skanowanie podatności**: *Nuclei* pozwala na skanowanie pod kątem różnego rodzaju podatności, od błędów konfiguracji serwera po luki w oprogramowaniu.
- **Szablony**: *Nuclei* korzysta z systemu szablonów do skanowania podatności. Szablony są zdefiniowanymi strukturami, które opisują specyficzne typy podatności i sposób, w jaki je zidentyfikować. Dzięki temu *Nuclei* jest elastyczne i może być dostosowane do różnych potrzeb.
- **Automatyzacja**: *Nuclei* można zintegrować z innymi narzędziami i procesami w celu automatyzacji skanowania podatności.
- **Współpraca społeczności**: Jako narzędzie *open-source*, *Nuclei* korzysta z wkładu społeczności bezpieczeństwa. To oznacza, że stale się rozwija i dostosowuje do nowych zagrożeń[41].
- **Szybkość**: *Nuclei* zostało zaprojektowane, aby działać szybko, co pozwala na skanowanie dużej ilości hostów w krótkim czasie.

Nuclei jest narzędziem przeznaczonym do wykorzystania przez bardziej zaawansowanych użytkowników. Posiada jedynie konsolowy interfejs, którego funkcje opierają się głównie na konfigurowaniu skanów. Narzędzie musi być uruchomione ręcznie. Pomimo faktu, że jest to narzędzie do testowania aplikacji webowych, potrafi również znajdować niektóre podatności poza tym zakresem, np. otwarte porty baz danych na hoście, na którym działa aplikacja.

2.7.4 WPscan

WPScan to narzędzie przeznaczone do skanowania stron internetowych opartych na systemie CMS *WordPress* pod kątem potencjalnych słabości obecnych w jego konkretnych instancjach [35]. Jest to jedno z bardziej znanych narzędzi penetracyjnych, z uwagi na to, że *WordPress* jest najpopularniejszym systemem do zarządzania treścią na świecie [42]. Oto niektóre z funkcji, które oferuje *WPScan*:

- **Wykrywanie wtyczek i motywów**: *WPScan* może skanować strony *WordPress* pod kątem używanych wtyczek i motywów, a następnie sprawdzać, czy są znane luki w tych komponentach.
- **Wykrywanie znanych podatności**: Autorzy projektu *WPscan* prowadzą bazę znanych podatności. Korzystając z niej, *WPscan* może znaleźć potencjalne luki w rdzeniu *WordPressa* lub w dodatkowych komponentach.
- **Enumeracja użytkowników**: *WPScan* może próbować wylistować nazwy użytkowników instancji *WordPress*, co może pomóc w atakach typu *brute-force* na hasła użytkowników.
- **Ataki siłowe**: *WPScan* może przeprowadzać ataki typu *brute-force* na konta użytkowników, próbując odgadnąć ich hasła.

- **Sprawdzanie aktualności komponentów**: *WPScan* może sprawdzić czy strona *WordPress* korzysta z najnowszej wersji rdzenia, wtyczek i motywów.

WPScan to narzędzie typu *open-source*, ale do wykorzystania pełnego potencjału warto korzystać z bazy podatności, która dostępna jest poprzez *API*. Darmowy plan pozwala na wykonanie 25 zapytań do *API* dziennie.

2.8 Ogólne porównanie narzędzi

Ta podsekcja zawiera kilka aspektów, które zostały omówione dla każdego ze skanerów. Uwydatnia to różnice pomiędzy nimi i sugeruje, w jakim kontekście warto lub nie warto użyć danego narzędzia.

2.8.1 Zakres działania

Nessus Professional: Skupia się na skanowaniu podatności infrastrukturalnych, skanując zarówno sprzęt, jak i oprogramowanie. Jest również używane do sprawdzania zgodności z regulacjami i standardami bezpieczeństwa.

Burp Suite Pro: Jest przeznaczony głównie do testowania bezpieczeństwa aplikacji webowych, oferując różne narzędzia, takie jak skaner podatności, proxy i inne narzędzia wspomagające testerów penetracyjnych.

Nuclei: Jest narzędziem do zautomatyzowanego skanowania podatności, które pozwala na sprawdzanie bezpieczeństwa aplikacji i serwisów internetowych.

WPscan: Jest narzędziem do skanowania podatności tylko dla CMSa *WordPress*.

2.8.2 Łatwość użycia

Nessus Professional: Posiada intuicyjny interfejs użytkownika, który ułatwia konfigurację skanów podatności. Może automatycznie generować szczegółowe raporty, co ułatwia analizę wyników. Dla osób bez doświadczenia w obszarze bezpieczeństwa sieci, konfiguracja i interpretacja wyników może być trudniejsza.

Burp Suite Pro: Dla doświadczonych testerów penetracyjnych, *Burp Suite Pro* jest bardzo użytecznym narzędziem. Ma wiele funkcji i narzędzi, które wymagają głębokiego zrozumienia bezpieczeństwa sieciowego. Dla osób bez specjalistycznej wiedzy, może być trudny do nauczenia. Ma dobrze zorganizowany interfejs użytkownika i wiele zasobów edukacyjnych dostępnych online, ale mnogość funkcji może być przytłaczająca dla użytkowników niezaznajomionych z aplikacją. Wersja *Burp Suite Enterprise* jest dużo łatwiejsza do obsługi, posiada czytelniejszy interfejs i nie uwzględnia funkcji, które nie są potrzebne do przeprowadzania automatycznych skanów.

Nuclei: Jako narzędzie przeznaczone dla specjalistów, może być trudniejsze do użycia dla osób niezaznajomionych z obsługą linii poleceń. Z drugiej strony *Nuclei* po zainstalowaniu można uruchomić jedną komendą z jednym argumentem (adresem skanowanej aplikacji webowej), co finalnie jest dużo szybszym sposobem na uruchomienie narzędzia niż tworzenie projektów, tak jak w przypadku *Nessus* czy *Burp Suite Pro*. Dla osób z doświadczeniem w skryptach i narzędziach typu *command-line*, *Nuclei* może być bardzo użytecznym narzędziem.

WPscan: Podobnie jak *Nuclei*, jest to narzędzie konsolowe, ale po poprawnej instalacji można uruchomić skanowanie jedną komendą. Wyniki są domyślnie zwracane w formacie JSON, co może być pomocne w parsowaniu wyników przez inne narzędzia.

2.8.3 Cena

- **Nessus Professional**: Jest to komercyjna wersja *Nessus*, która jest dostępna za opłatą. Bazując na cenie podanej na stronie internetowej *Tenable* w momencie pisania pracy, licencja roczna wynosi niecałe 19 000zł.
- **Burp Suite Pro**: Jest to płatne narzędzie, z licencjonowaniem rocznym. Kosztuje 449 euro za użytkownika. Wersja *Enterprise* to koszt od 8395 euro rocznie.
- **Nuclei**: Jako narzędzie *open-source*, jest dostępne za darmo.
- **WPscan**: Jako narzędzie *open-source*, jest dostępne za darmo. Darmowy dostęp do bazy podatności *WPscan* ogranicza się do 25 zapytań dziennie. Na stronie internetowej `wpscan.com` nie ma informacji o cenie w przypadku wyższego zapotrzebowania.

2.8.4 Rozszerzalność

Nessus Professional: Oferuje szerokie możliwości konfiguracji i dostosowywania skanów, ale możliwości rozwoju są ograniczone w porównaniu do otwartych narzędzi typu *open-source*. *Nessus* pozwala na tworzenie i importowanie niestandardowych plików `.audit`, które mogą być używane do tworzenia niestandardowych skanów konfiguracji. Nie oferuje jednak bezpośredniego interfejsu do tworzenia wtyczek lub rozszerzeń, choć jego funkcjonalność może być rozszerzona poprzez *API*, które pozwala na integrację z innymi narzędziami i automatyzację niektórych funkcji.

Burp Suite Pro: Ma bardzo rozbudowany ekosystem rozszerzeń, które pozwalają na znaczne rozszerzenie funkcjonalności narzędzia. *Burp Extensions* mogą być tworzone w różnych językach programowania, takich jak *Java*, *Python* i *Ruby*, a następnie importowane do *Burp Suite*. Wiele tych rozszerzeń jest dostępnych w *BApp Store*, dostarczanym przez *PortSwigger*.

Nuclei: Jako narzędzie *open-source*, *Nuclei* oferuje znaczne możliwości rozwoju. Kluczową cechą *Nuclei* jest system szablonów, który pozwala użytkownikom tworzyć własne szablony skanowania, które mogą być dostosowane do różnych typów podatności. Społeczność *Nuclei* aktywnie tworzy i udostępnia szablony, które mogą być łatwo dodane do narzędzia. W dodatku, *Nuclei* oferuje API, które może być używane do integracji z innymi narzędziami i automatyzacji skanowania.

WPscan: Narzędzie może być modyfikowane i ulepszane dzięki temu, że jego kod źródłowy jest publicznie znany. *WPscan* nie ma jednak możliwości tworzenia własnych reguł czy szablonów podatności. Należy jednak zauważyć, że dzięki utrzymywanej bazie podatności aplikacja jest zawsze aktualna.

2.8.5 Automatyzacja

Nessus Professional: Oferuje rozbudowane możliwości automatyzacji. Użytkownicy mogą zaplanować skany podatności, które uruchamiają się automatycznie w wyznaczonym czasie. Możliwe jest również automatyczne powiadamianie o wynikach skanowania przez e-mail. *Nessus* oferuje również *REST API*, które można wykorzystać do automatyzacji wielu aspektów zarządzania skanami w tym tworzenia, modyfikowania, uruchamiania i usuwania skanów, a także pobierania i analizowania wyników skanowania.

Burp Suite Pro: Oferuje różne funkcje automatyzacji, szczególnie w przypadku skanera podatności. Można zautomatyzować skanowanie i testowanie różnych aspektów aplikacji webowej. *Burp Suite Pro* oferuje również możliwość tworzenia i zarządzania zadaniami skanowania, które mogą być uruchamiane w tle. Ponadto, dzięki *Burp Extender*, można tworzyć

i używać rozszerzeń, które dodają nowe funkcje lub modyfikują istniejące, co potencjalnie zwiększa możliwości automatyzacji. Wersja Pro nie posiada jednak wbudowanych funkcji dotyczących monitorowania hostów i uruchamiania skanów np. w określonych dniach, co oferuje wersja *Enterprise*.

Nuclei: Jest zaprojektowane z myślą o automatyzacji, ale samo w sobie nie posiada takich funkcji. *Nuclei* może zostać *wcielone* w inne narzędzie, które uruchamiałoby skany, bądź może być wdrożone w proces CI/CD aplikacji [43].

WPscan: Również nie posiada funkcji z myślą o automatyzacji, jednakże dzięki otwartości projektu istnieją zewnętrzne narzędzia (np. wpwatcher [44]), które pozwalają na jego automatyzację.

2.9 Porównanie narzędzi na podstawie rezultatów

Temat kompleksowego porównania skanerów i wybrania najskuteczniejszych skanerów jest zbyt szeroki, aby mógł być częścią tej pracy, jednakże takie porównania można znaleźć w innych pracach np. [45], [46], [47]. Przedstawione porównanie ma na celu zwrócenie uwagi na różnice między narzędziami.

2.9.1 Metodyka

Porównanie dotyczy prób wykonanych na instancji *DVWP (Damn Vulnerable WordPress)* z istniejącymi podatnościami [37]:

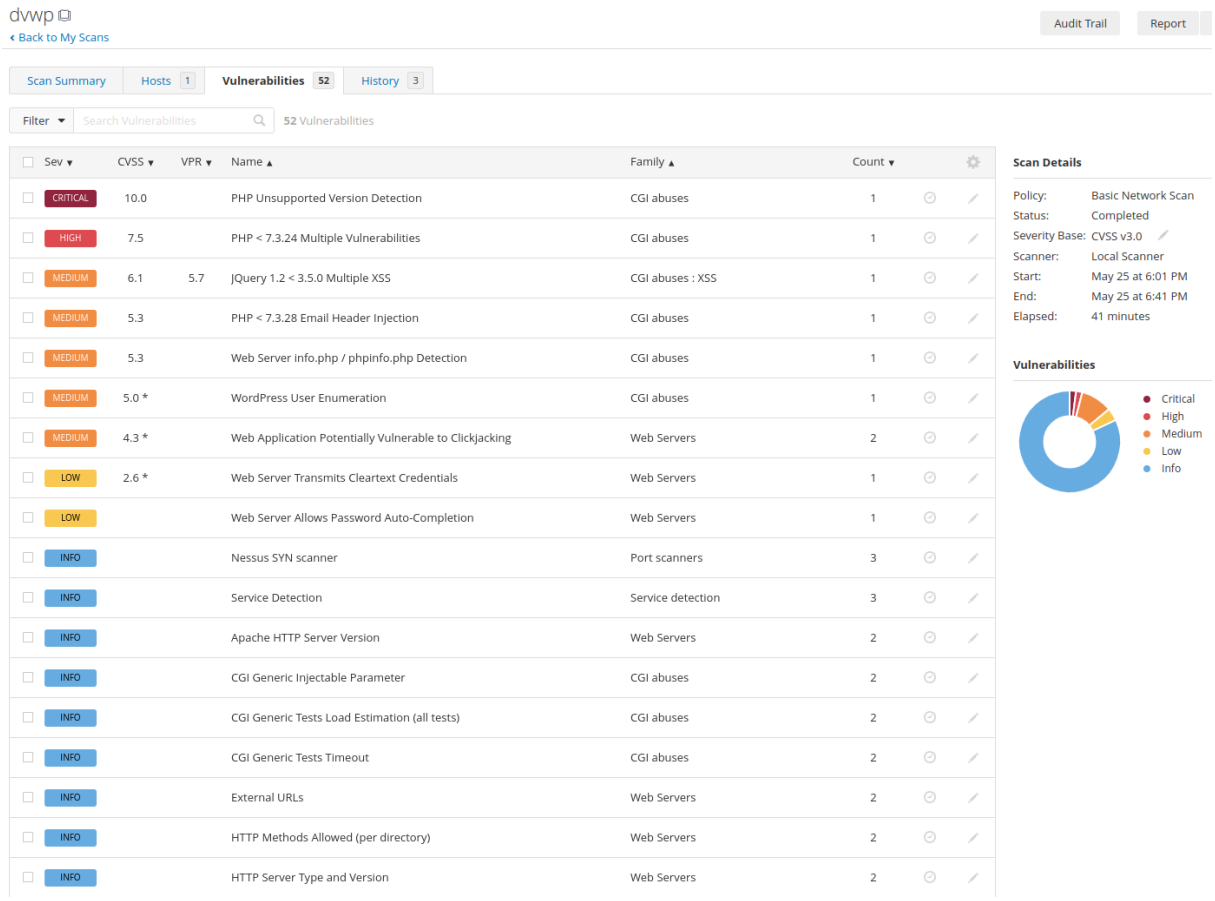
1. Ominięcie uwierzytelnienia
2. Przeglądanie katalogów prowadzące do zdalnego wykonania kodu
3. Nieautoryzowany dostęp do bazy danych i zdalne wykonanie kodu
4. Nieautoryzowana modyfikacja ustawień
5. Listowanie katalogów
6. Wyświetlanie błędów PHP
7. Ujawnienie informacji poprzez funkcję phpinfo
8. Ujawnienie informacji poprzez publicznie dostępną kopię bazy danych
9. Publicznie dostępne oprogramowanie Adminer
10. Publicznie dostępne oprogramowanie Search-Replace-DB
11. Niepoprawna polityka CORS (Cross-origin Resource Sharing)
12. Użycie nieaktualnych komponentów ze znanymi podatnościami

Możliwe, że nie wszystkie błędy zostały wymienione. Szczegóły dotyczące uruchomienia *DVWP* są opisane w rozdziale 5.

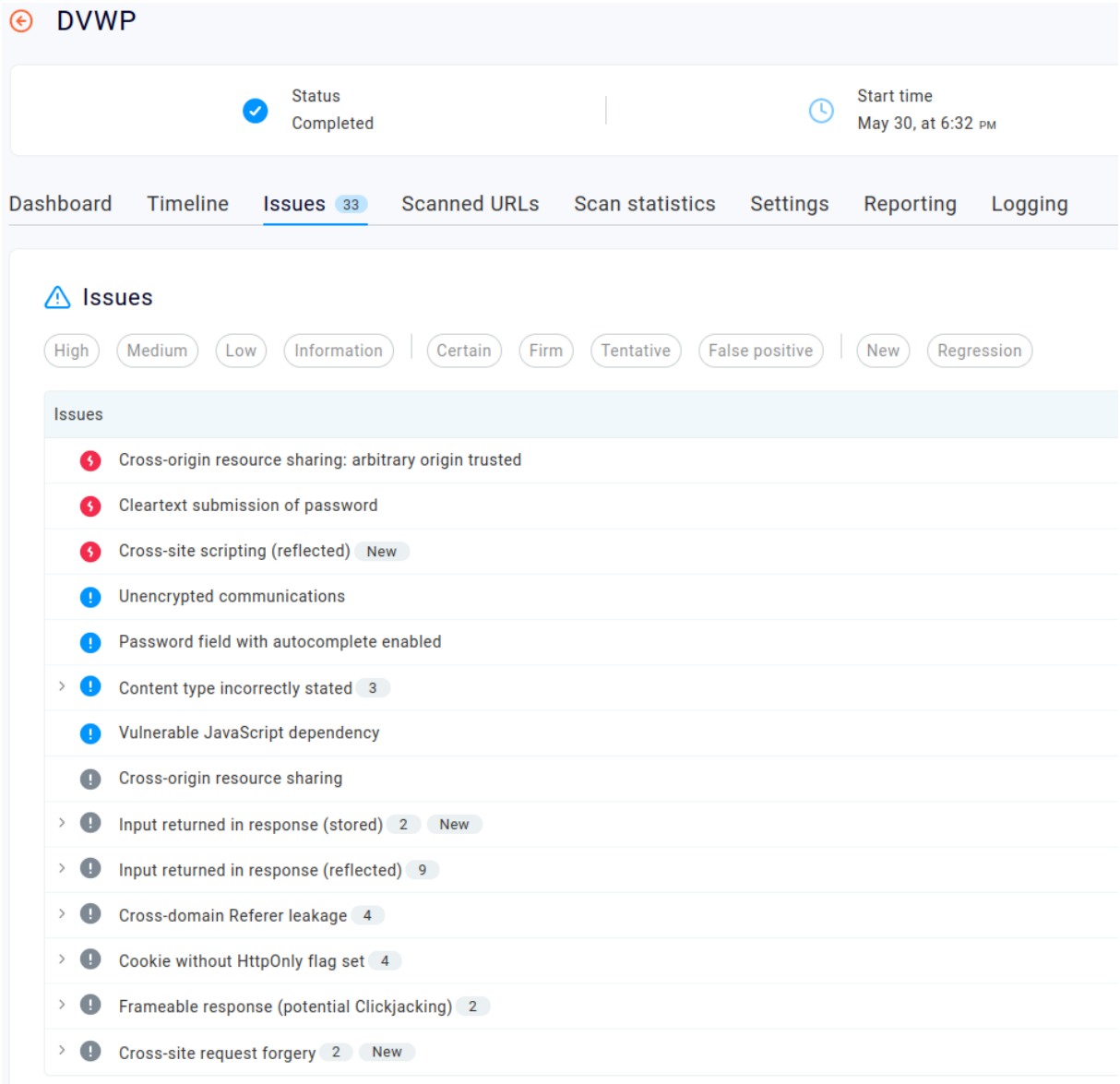
2.9.2 Wyniki

2.9.2.1 Nessus Professional

Uruchomiono standardowy skan (*Basic Network Scan*) na instancji *DVWP*. Łącznie wykryto 38 podatności: 1 krytyczną, 1 wysoką, 5 średnich i 2 niskie. Pozostałe podatności były oznaczone jako informacyjne. Skan trwał 40 minut. *Nessus* nie wykrył najistotniejszych pod kątem bezpieczeństwa podatności. Rysunek 2.1 prezentuje pełny wynik.



Rysunek 2.1: Wyniki skanera Nessus



Rysunek 2.2: Wyniki skanera *Burp Suite Enterprise*

2.9.2.2 Burp Suite Enterprise

Podjęto próby uruchomienia różnych skanów, ale skany bardziej skomplikowane niż *Fast* trwały ponad dzień i dalej się nie kończyły. Przedstawione wyniki dotyczą zatem skanu w konfiguracji *Fast*. Całość skanu zajęła 11 godzin i 14 minut. Łącznie wykryto 33 podatności: 3 wysokie i 6 niskich. Pozostałe podatności były oznaczone jako informacyjne.

Podatność `Reflected Cross-site Scripting` była fałszywym alarmem (*False Positive*), ponieważ niektóre tagi HTML mogą być zawartością komentarza, ale nie stanowią one ryzyka. Przedstawiony w *Burp Suite Enterprise* dowód koncepcji (*Proof of Concept, PoC*) ograniczał się do wskazania, ze możliwe jest dodanie tagu `<a>` w komentarzu. Rysunek 2.2 prezentuje pełny wynik.

2.9.2.3 Nuclei

Użyto *Nuclei* ze standardową konfiguracją. *Nuclei* nie podaje czasu trwania skanu, ale trwał kilka minut. Łącznie wykryto 37 podatności: 1 krytyczną, 1 wysoką, 2 średnie i 1 niską.

Pozostałe podatności były oznaczone jako informacyjne. *Nuclei* znalazł część podatności w aplikacji. Listing 2.1 zawiera pełny wynik.

Listing 2.1: Wynik skanera Nuclei

```
$ \textit{Nuclei} -u http://64.176.67.249:31337

                     __     _
   _____  __  _______/ /__  (_)
  / __ \/ / / / ___/ / _ \/ /
 / / / / /_/ / /__/ /  __/ /
/_/ /_/\__,_/\___/_/\___/_/   v2.9.4

                projectdiscovery.io

[INF] Current \textit{Nuclei} version: v2.9.4 (latest)
[INF] Current nuclei-templates version: 9.5.0 (latest)
[INF] New templates added in latest release: 62
[INF] Templates loaded for current scan: 5958
[INF] Targets loaded for current scan: 1
[INF] Templates clustered: 1057 (Reduced 999 Requests)
[php-detect] [http] [info] http://64.176.67.249:31337 [7.1.33]
[apache-detect] [http] [info] http://64.176.67.249:31337 [Apache/2.4.38 ↙
   ↳ (Debian)]
[tech-detect:php] [http] [info] http://64.176.67.249:31337
[metatag-cms] [http] [info] http://64.176.67.249:31337 ↙
   ↳ [\textit{WordPress} 5.3]
[http-missing-security-headers:x-frame-options] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:referrer-policy] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:access-control-expose-headers] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:access-control-allow-headers] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:cross-origin-opener-policy] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:cross-origin-resource-policy] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:access-control-allow-methods] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:content-security-policy] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:permissions-policy] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:x-content-type-options] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:clear-site-data] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:access-control-allow-credentials] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:strict-transport-security] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:x-permitted-cross-domain-policies] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:cross-origin-embedder-policy] [http] ↙
   ↳ [info] http://64.176.67.249:31337
[http-missing-security-headers:access-control-allow-origin] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[http-missing-security-headers:access-control-max-age] [http] [info] ↙
   ↳ http://64.176.67.249:31337
[INF] Using Interactsh Server: oast.site
[WRN] [wordpress-iwp-client] Malformed version: trunk
```

```
[wordpress-iwp-client:detected_version] [http] [info] ↙
    ↳ http://64.176.67.249:31337 ↙
    ↳ /wp-content/plugins/iwp-client/readme.txt [trunk] ↙
    ↳ [last_version="trunk"]
[CVE-2017-5487:usernames] [http] [medium] ↙
    ↳ http://64.176.67.249:31337/?rest_route=/wp/v2/users/ [admin]
[adminer-panel] [http] [info] http://64.176.67.249:31337/adminer.php ↙
    ↳ [4.6.2]
[openssh-detect] [tcp] [info] 64.176.67.249:22 [SSH-2.0-OpenSSH_9.0p1 ↙
    ↳ Ubuntu-1ubuntu8]
[phpinfo-files] [http] [low] http://64.176.67.249:31337/info.php [7.1.33]
[wordpress-detect:version_by_js] [http] [info] http://64.176.67.249:31337 ↙
    ↳ [5.3]
[waf-detect:apachegeneric] [http] [info] http://64.176.67.249:31337/
[wordpress-xmlrpc-file] [http] [info] http://64.176.67.249:31337/xmlrpc.php
[ptr-fingerprint] [dns] [info] 64.176.67.249 ↙
    ↳ [64.176.67.249.vultrusercontent.com.]
[CVE-2020-8772] [http] [critical] http://64.176.67.249:31337/
[wp-license-file] [http] [info] http://64.176.67.249:31337/license.txt
[wordpress-login] [http] [info] http://64.176.67.249:31337/wp-login.php
[wordpress-readme-file] [http] [info] ↙
    ↳ http://64.176.67.249:31337/readme.html
[default-sql-dump] [http] [medium] http://64.176.67.249:31337/dump.sql
[wp-xmlrpc-pingback-detection] [http] [info] ↙
    ↳ http://64.176.67.249:31337/xmlrpc.php
[CVE-2021-21311] [http] [high] http://64.176.67.249:31337/adminer.php ↙
    ↳ ?elastic=example.org&username=i4bmsofv&db=u3pzmzxj ↙
    ↳ [path="/adminer.php"]
```

2.9.2.4 WPscan

Użyto *WPscan* w konfiguracji z następującymi zmianami: `-random-user-agent -f json -api-token <token>`. Najistotniejszą zmianą jest dodanie tokenu do *API*, który pozwala na skorzystanie z bazy podatności *WPscan*. *WPscan* nie zwraca informacji o ryzyku podatności, zamiast tego zwraca informacje takie jak numer *CVE*, linki do opisów podatności itp. Łącznie zidentyfikowano 58 podatności oraz zwrócono dodatkowe informacje na temat instancji WordPress. Skan trwał 5 sekund. *WPscan* znalazł najwięcej podatności ze wszystkich skanerów. Listing 2.1 zawiera jedynie tytuły znalezionych podatności, ze względu na zbyt obszerny wynik skanera w formie tekstowej.

Listing 2.2: Tytuły podatności znalezionych przez WPscan

```
Social Warfare <= 3.5.2 - Unauthenticated Arbitrary Settings Update
Social Warfare <= 3.5.2 - Unauthenticated Remote Code Execution (RCE)
Social Warfare < 4.3.1 - Subscriber+ Post Meta Deletion
Social Warfare < 4.4.0 - Post Meta Deletion via CSRF
WP Advanced Search < 3.3.4 - Unauthenticated Database Access and Remote ↙
    ↳ Code Execution (RCE)
WP Advanced Search < 3.3.6 - Unauthenticated SQL Injection
WP-Advanced-Search < 3.3.7 - Authenticated SQL Injection
WP-Advanced-Search <= 3.3.8 - Settings Update via CSRF
\textit{WordPress} File Upload < 4.13.0 - Directory Traversal to RCE
\textit{WordPress} File Upload < 4.16.3 - Contributor+ Stored Cross-Site ↙
    ↳ Scripting via Shortcode
\textit{WordPress} File Upload < 4.16.3 - Contributor+ Stored Cross-Site ↙
    ↳ Scripting via Malicious SVG
\textit{WordPress} File Upload < 4.16.3 - Contributor+ Path Traversal to ↙
    ↳ RCE
\textit{WordPress} <= 5.3 - Authenticated Improper Access Controls in ↙
    ↳ REST API
\textit{WordPress} <= 5.3 - Authenticated Stored XSS via Crafted Links
```

```
\textit{WordPress} <= 5.3 - Authenticated Stored XSS via Block Editor ↙
    ↳ Content
\textit{WordPress} <= 5.3 - wp_kses_bad_protocol() Colon Bypass
\textit{WordPress} < 5.4.1 - Password Reset Tokens Failed to Be Properly ↙
    ↳ Invalidated
\textit{WordPress} < 5.4.1 - Unauthenticated Users View Private Posts
\textit{WordPress} < 5.4.1 - Authenticated Cross-Site Scripting (XSS) in ↙
    ↳ Customizer
\textit{WordPress} < 5.4.1 - Authenticated Cross-Site Scripting (XSS) in ↙
    ↳ Search Block
\textit{WordPress} < 5.4.1 - Cross-Site Scripting (XSS) in wp-object-cache
\textit{WordPress} < 5.4.1 - Authenticated Cross-Site Scripting (XSS) in ↙
    ↳ File Uploads
\textit{WordPress} < 5.4.2 - Authenticated XSS in Block Editor
\textit{WordPress} < 5.4.2 - Authenticated XSS via Media Files
\textit{WordPress} < 5.4.2 - Open Redirection
\textit{WordPress} < 5.4.2 - Authenticated Stored XSS via Theme Upload
\textit{WordPress} < 5.4.2 - Misuse of set-screen-option Leading to ↙
    ↳ Privilege Escalation
\textit{WordPress} < 5.4.2 - Disclosure of Password-Protected Page/Post ↙
    ↳ Comments
\textit{WordPress} 4.7-5.7 - Authenticated Password Protected Pages ↙
    ↳ Exposure
\textit{WordPress} 3.7 to 5.7.1 - Object Injection in PHPMailer
\textit{WordPress} < 5.8.2 - Expired DST Root CA X3 Certificate
\textit{WordPress} < 5.8 - Plugin Confusion
\textit{WordPress} < 5.8.3 - SQL Injection via WP_Query
\textit{WordPress} < 5.8.3 - Author+ Stored XSS via Post Slugs
\textit{WordPress} 4.1-5.8.2 - SQL Injection via WP_Meta_Query
\textit{WordPress} < 5.8.3 - Super Admin Object Injection in Multisites
\textit{WordPress} < 5.9.2 - Prototype Pollution in jQuery
WP < 6.0.2 - Reflected Cross-Site Scripting
WP < 6.0.2 - Authenticated Stored Cross-Site Scripting
WP < 6.0.2 - SQLi via Link API
WP < 6.0.3 - Stored XSS via wp-mail.php
WP < 6.0.3 - Open Redirect via wp_nonce_ays
WP < 6.0.3 - Email Address Disclosure via wp-mail.php
WP < 6.0.3 - Reflected XSS via SQLi in Media Library
WP < 6.0.3 - CSRF in wp-trackback.php
WP < 6.0.3 - Stored XSS via the Customizer
WP < 6.0.3 - Stored XSS via Comment Editing
WP < 6.0.3 - Content from Multipart Emails Leaked
WP < 6.0.3 - SQLi in WP_Date_Query
WP < 6.0.3 - Stored XSS via RSS Widget
WP < 6.0.3 - Data Exposure via REST Terms/Tags Endpoint
WP < 6.0.3 - Multiple Stored XSS via Gutenberg
WP <= 6.2 - Unauthenticated Blind SSRF via DNS Rebinding
WP < 6.2.1 - Directory Traversal via Translation Files
WP < 6.2.1 - Thumbnail Image Update via CSRF
WP < 6.2.1 - Contributor+ Stored XSS via Open Embed Auto Discovery
WP < 6.2.2 - Shortcode Execution in User Generated Data
WP < 6.2.1 - Contributor+ Content Injection
```

2.9.3 Wnioski

Każde z narzędzi i skanerów przedstawionych wyżej dało różniące się rezultaty, pomimo tego, że testowane były na tej samej aplikacji. Najbardziej precyzyjne rezultaty dał *WPscan*, czego można było się spodziewać, ponieważ jest to narzędzie dedykowane konkretnie dla platformy *WordPress*. *WPscan* wykrywa dotychczas znane problemy, czyli takie, które już zostały zgłoszone i najczęściej są poprawione. Podobnie działa *Nuclei*, ponieważ bazuje na konkretnych szablonach podatności. *Burp Suite Enterprise* zaprezentował odmienne zachowanie, ponieważ nie odnalazł żadnej ze znanych podatności, ale próbował odnaleźć nowe

(np. *Cross-site scripting*). Potwierdza to także największa ilość wykonanych zapytań. Z kolei *Nessus Professional* skupił się na skanowaniu aspektów w dużym stopniu pomijających warstwę aplikacji [48]. Podsumowanie wniosków przedstawia Tabela 2.1.

Tabela 2.1: Porównanie skanerów na przykładzie DVWP

Narzędzie	**Przeznaczenie skanera**	**Skuteczność na hoście**	**Skuteczność w DVWP**
Nessus Pro	infrastruktura	wysoka	średnia
Burp Suite EE	aplikacje webowe	średnia	niska
Nuclei	aplikacje webowe	wysoka	wysoka
wpscan	instancje WordPress	niska	bardzo wysoka

Ogólnym wnioskiem na podstawie obserwacji jest to, że ciężko opierać bezpieczeństwo aplikacji webowych na jednym skanerze. Nawet skanując system dedykowanym skanerem (w tym przypadku *WPscan*) nie zostaną odnalezione podatności poboczne, takie jak konfiguracja hosta czy publicznie dostępne pliki z kopiami zapasowymi, czyli podatności nie będące częścią rdzenia *WordPress*, publicznych motywów i publicznych pluginów.

3. Generyczny skaner bezpieczeństwa

Opierając się na wadach i zaletach różnych narzędzi oraz wnioskach przedstawionych w poprzednim rozdziale, stworzono koncepcję generycznego skanera bezpieczeństwa, który może rozwiązać część wskazanych problemów. Dalsza część tego rozdziału zawiera opis rozwiązania wraz z głównymi założeniami koncepcyjnymi. Na koniec zebrano strategie pozwalające na uniknięcie zidentyfikowanych problemów.

3.1 Ogólny opis rozwiązania

Założeniem generycznego skanera bezpieczeństwa jest stworzenie narzędzia mogącego działać w różnych środowiskach i skanować wiele systemów. W przeciwieństwie do wielu komercyjnych skanerów na rynku, taki skaner ma wykorzystywać już istniejące rozwiązania i ma być rozszerzalny poprzez możliwość dopisywania wtyczek.

Generyczny skaner ma rozwiązać potrzebę manualnego korzystania z wielu narzędzi w celu weryfikacji zabezpieczeń. Dodatkowo, skaner powinien uruchamiać odpowiednie narzędzia w zależności od usługi działającej na danym hoście.

Najbardziej istotnym aspektem generycznego skanera bezpieczeństwa jest modularność, czyli możliwość dodawania modułów skanujących. Programy modułowe pozwalają na rozszerzanie ich funkcji poprzez mechanizm dodawania wtyczek (*pluginów*).

Koncepcja powstała z założeniem, że nie jest możliwe samodzielne utworzenie skanera, który weryfikowałby wszystko i znajdowałby każdą podatność - cyberbezpieczeństwo jest tak szerokim obszarem, że nie da się specjalizować w jego każdym dziale. Mnogość rozwiązań (np. wiele silników forów internetowych) także wymaga wiedzy w zakresie sposobu budowy konkretnych aplikacji. Modularność generycznego skanera bezpieczeństwa pozwala dowolnej osobie na dodanie wtyczki, która będzie zawierać kolejne kontrolki (*checki*). Dzięki temu jedno narzędzie jest w stanie wykonać weryfikację wielu różnych aspektów bezpieczeństwa, a te pominięte mogą być wprowadzone samodzielnie.

Aby zapewnić ciągłość rozwoju, narzędzie musi być mieć otwarty kod źródłowy (*open-source*).

3.2 Podział skanera

Skaner składa się z dwóch części - rdzenia i wtyczek. Rdzeń odpowiada za zarządzanie informacjami i ogólnymi procesami (np. uruchamianiem skanów), a wtyczki zajmują się przeprowadzaniem skanów w celu identyfikacji podatności, oraz późniejszym przygotowaniem rezultatów do zwrócenia w odpowiedniej formie.

3.2.1 Rdzeń

Z uwagi na modułowy charakter rozwiązania, rdzeń skanera powinien zawierać wszystkie funkcje, których mogą wymagać użytkownicy, ze względu na to, że skaner ma być przeznaczony do działania w różnych miejscach i zastosowaniach. Najważniejsze funkcjonalności, które powinny się znaleźć w rdzeniu, to:

- **Zarządzanie hostami**: Dodawanie, usuwanie i edytowanie hostów, które mają być skanowane,
- **Zarządzanie konfiguracjami skanów**: Dodawanie, usuwanie i edytowanie konfiguracji, według których będą uruchamiane skany,

- **Uruchamianie skanów**: Wykonywanie skanowania na podstawie wspomnianej wyżej konfiguracji,
- **Monitorowanie skanów**: Z uwagi na czasochłonność operacji skanowania, niezbędne jest dostarczanie informacji o obecnym etapie skanowania (np. % wykonanych zadań),
- **Przeglądanie rezultatów**: Rezultaty skanowania muszą być zapisywane, a użytkownik ma mieć do nich dostęp,
- **Komunikacja z wtyczkami**: Ze względu na niezależność wtyczek, muszą one komunikować się z rdzeniem, np. w celu informowania o obecnym statusie skanowania lub przesłania wyników skanu,
- **Planowanie skanów**: Cykliczne uruchamianie skanów wyręczyłoby człowieka z konieczności ręcznego uruchamiania skanu w określonych interwałach.

Z punktu widzenia skanowania podatności, rdzeń sam w sobie nie stanowi istotnej funkcji, ponieważ służy głównie do zarządzania. Z drugiej strony, rdzeń gromadzi wiele informacji, które są danymi wejściowymi do wtyczek (w najprostszym przypadku będzie to sama nazwa hosta). Dane wyjściowe z jednej wtyczki mogą być używane jako dane wejściowe drugiej. Można zatem powiedzieć, że wtyczki są narzędziami używanymi przez rdzeń.

Kolejnym założeniem jest dostarczanie przez rdzeń funkcjonalności w sposób bliski doskonałości, tzn. aby użytkownicy mogli na nim polegać i nie musieli wdrażać własnych zmian. Niestabilność jednej z wtyczek nie powinna być problemem dla całego skanu, ale takie wyjątki muszą być obsłużone przez rdzeń. Niedopuszczalna jest jednak sytuacja, w której rdzeń przestałby działać, ponieważ skutecznie uniemożliwiłoby to korzystanie z całego rozwiązania. Kolejnym powodem, aby rdzeń był najbardziej dopracowaną częścią rozwiązania, jest zmniejszenie progu wejścia dla osób wnoszących wkład w projekt. W idealnej sytuacji zmiany w rdzeniu nie byłyby konieczne, a osoby wnoszące wkład wprowadzałyby ulepszenia jedynie we wtyczkach. Próg wejścia w modyfikację wtyczki jest dużo niższy, ponieważ nie wymaga znajomości architektury całego rozwiązania, a jedynie sposobu uruchamiania i działania wtyczki we współpracy z rdzeniem, który to kontroluje.

3.2.2 Wtyczki

Wtyczki mają zapewniać funkcje związane z wyszukiwaniem podatności w systemach informatycznych. Każda wtyczka (z pominięciem wyjątku opisanego później) ma mieć możliwie niezależną formę, a w większości sytuacji kod wtyczki ma służyć do uruchamiania i zarządzania zewnętrznym procesem, którym jest inne narzędzie.

Ze względu na charakter działania, wtyczki są podzielone na dwa typy: rekonesans i skan.

3.2.2.1 Rekonesans (discovery)

Rekonesans, w kontekście cyberbezpieczeństwa, to pierwszy etap w większości cyberataków, którego celem jest zebranie informacji o celu ataku. Te informacje mogą obejmować zarówno aspekty technologiczne, takie jak używany sprzęt czy wersje komponentów oprogramowania, jak i dane dotyczące fizycznej lokalizacji ofiary, numery telefonów, nazwiska osób pracujących w docelowej organizacji i ich adresy e-mail. Każdy kawałek wiedzy może zostać wykorzystany do opracowania *exploitu* lub do ujawnienia słabości w systemach obronnych [49].

Rekonesans zazwyczaj polega na złożonym zestawie technik i procesów. Można go podzielić na cztery główne klasy technik:

- **Inżynieria społeczna**: Metody zbierania informacji mające na celu oszukanie osoby lub skłonienie jej do określonego zachowania.

- **Wywiad internetowy**: Metody wykorzystujące informacje publicznie dostępne w Internecie, w tym bazy danych dostępne za pośrednictwem sieci Web.
- **Zbieranie informacji o sieci**: Metody mapowania infrastruktury sieciowej (lub komputerowej) ofiary.
- **Kanały boczne (*side-channels*)**: Metody wykorzystujące niezamierzone wycieki informacji ze strony ofiary.

W skrócie, celem rekonesansu jest zrozumienie jak system jest skonfigurowany, jakie ma potencjalne słabości oraz jakie są potencjalne wektory ataku. Te informacje mogą potem być wykorzystane do planowania i przeprowadzania ataku.

Wtyczki typu *discovery* mają zatem służyć do rozpoznania skanowanego hosta i zebrania informacji, które mogą być później użyte przez inne wtyczki. Wracając do opisanego wcześniej wyjątku niezależności wtyczek od rdzenia - założenie skanera jest takie, że wtyczki *discovery* mogą wymagać konkretnych zmian w kodzie rdzenia, aby w pełni funkcjonowały. Przykładem może być zbieranie informacji o domenach zmapowanych na dany host: w takim przypadku konieczna by była dodatkowa kolumna w bazie bądź oddzielna tabela. Tym samym kod aplikacji również musiałby być zmodyfikowany.

3.2.2.2 Skan (Scan)

Wtyczki do skanowania są w pełni niezależne od siebie, ale mogą wymagać danych zebranych przez wtyczki typu *discovery*. Skuteczność całego rozwiązania jest uzależniona od ilości i jakości napisanych wtyczek. Tak jak wspomniano wcześniej, wtyczki to często jedynie osłona (*wrapper*) na zewnętrzny program, który jest zarządzany przez wtyczkę.

Wtyczki mają określony cykl życia:

- **Utworzenie**: Obiekt wtyczki jest tworzony, argumenty są przekazane, ale skan się jeszcze nie zaczął.
- **Uruchomienie**: Wtyczka rozpoczyna skanowanie, a rezultaty są na bieżąco zapisywane. Dodatkowo wtyczka może zwracać aktualny postęp (0-100%).
- **Zakończenie**: Wtyczka przesyła informację o tym, że skończyła pracę.
- **Przygotowanie listy podatności**: Na żądanie rdzenia wtyczka parsuje wynik skanowania i zwraca listę znalezionych podatności do rdzenia.

3.3 Główne elementy rozwiązania

Elementy będą opisywane od największego, do najmniejszego (mniejsze elementy *zawierają się* w większych).

Projekt

Projekt ma na celu grupowanie *Celów (Targets)*. Jedyna właściwość projektu to nazwa, która ma ułatwiać wyszukiwanie grup *Celów* w przypadkach, w których istnieje dużo *Projektów*. Może to być np. *Serwery szkolne*.

Cel (Target)

Cel to host, który ma następujące główne właściwości: nazwa (np. *Strona PJATK*), IP/domena (np. *pja.edu.pl*) i lista portów. Jest to prawdopodobnie najbardziej istotny element dla użytkownika, ponieważ skany są uruchamiane na konkretnych *Celach*.

Grupa skanów

To abstrakcyjna grupa kilku *Skanów*. Jej istnienie jest potrzebne jedynie w celu intuicyjnego wyświetlania użytkownikowi uruchomionych działań w ramach jednej *Konfiguracji skanu*. Nie ma to jednak żadnego wpływu na ogólną skuteczność skanowania.

Skan (Scan)

Skan to uruchomienie wtyczki na pojedynczym porcie *Celu*. Każdy skan zwraca własną informację o postępie i rezultatach do rdzenia. W przypadku wystąpienia problemów ze skanowaniem, nie dojdzie do efektu kaskadowego - zamiast tego ten jeden skan się nie powiedzie, a reszta będzie działała prawidłowo, ponieważ skany są niezależne od siebie.

Podatność

Znaleziona podczas wykonywania *Skanu* luka bezpieczeństwa.

3.4 Metoda działania

Poniżej znajduje się podstawowy schemat działania skanera. Pod listą każdy z elementów został opisany obszerniej.

1. **Stworzenie projektu**

   Należy stworzyć *Projekt*, aby móc grupować *Cele*. Każdy *Cel* musi być przypisany do Projektu. Docelowo *Cele* w projekcie mają być częściowo powiązane, np. działać w jednej firmie. Dzięki temu możliwe będzie tworzenie statystyk dla całego projektu.

2. **Dodanie hosta (*Celu*) w projekcie w formie adresu IP lub domeny**

   Dodawanie *Celu* polega na wpisaniu jego adresu IP lub domeny. Dodatkowo można podać zakres portów, które będą skanowane.

3. **Dodanie konfiguracji skanu**

   Polega to na wybraniu wtyczek, które mają być użyte na określonych *Celach*. Najprostszym scenariuszem jest użycie wszystkich wtyczek na wszystkich hostach, ale nie zawsze jest to optymalne rozwiązanie. Przykładem mogą być sytuacje wymagające użycia jak najmniej inwazyjnych wtyczek, aby zlikwidować możliwość ataków typu "Odmowa dostępu do usługi" (*Denial of Service, DoS*) na własne aplikacje.

4. **Uruchomienie skanu rozpoznającego działające serwisy na hostach**

   Uruchamiany jest skan rozpoznawczy (*discovery*), który wyszukuje otwarte porty. W późniejszym etapie wtyczki będą uruchamiane tylko na nich. Dodatkowo, na portach identyfikowane są działające serwisy, aby uruchamiane były wtyczki właściwe dla usługi.

5. **Uruchomienie głównego skanera**

   Główny skaner jest uruchamiany na podstawie konfiguracji skanu utworzonej wcześniej. Wykorzystywane są wszystkie wybrane wcześniej wtyczki, pomijając te, które nie są zgodne ze zidentyfikowanymi wcześniej usługami. Wtyczka jest uruchamiana dla każdego zgodnego portu na każdym hoście (czyli np. mając 3 hosty z 5 zgodnymi serwisami na każdym, to wtyczka zostanie uruchomiona 15 razy).

6. **Zbieranie rezultatów**

   Po każdym zakończonym użyciu wtyczki, parsuje ona rezultaty i tworzy z tego listę podatności, która jest zwracana do rdzenia. Ta akcja dzieje się po każdym użyciu wtyczki, zatem dane są zwracane na bieżąco. Znalezione podatności są dodawane do listy podatności określonego portu na określonym hoście w rdzeniu. Obiekt

podatności zwracany przez wtyczki zawiera opis, *Cel*, port, ryzyko oraz numer *CWE* (nieobowiązkowo).

Rysunek 3.1 prezentuje powyższy schemat w formie diagramu sekwencyjnego.

3.5 Problemy i rozwiązania

W trakcie tworzenia koncepcji oraz implementacji przemyślano wiele problemów, jakie mogą pojawić się (lub pojawiły się) w aplikacji. Ta sekcja opisuje te najistotniejsze wraz z rozwiązaniami. Na niektóre problemy nie można było znaleźć perfekcyjnego rozwiązania i w takim przypadku zostały opisane różne strategie wraz z ich zaletami i wadami.

3.5.1 Wykorzystanie wyników jednego narzędzia do uruchomienia następnego

Niektóre narzędzia nie działają z domyślną lub przygotowaną wcześniej konfiguracją. Zamiast tego mogą wymagać podania konkretnych danych wejściowych, np. konkretne punkty końcowe API. Takie informacje mogą być podane przez użytkownika w argumentach bądź plikach wejściowych, ale zaprzeczałoby to idei automatycznego skanowania.

Jedną z możliwości rozwiązania problemu byłoby uzależnianie jednego narzędzia od drugiego. Wtedy wskazane byłoby wprowadzenie dodatkowych zmian:

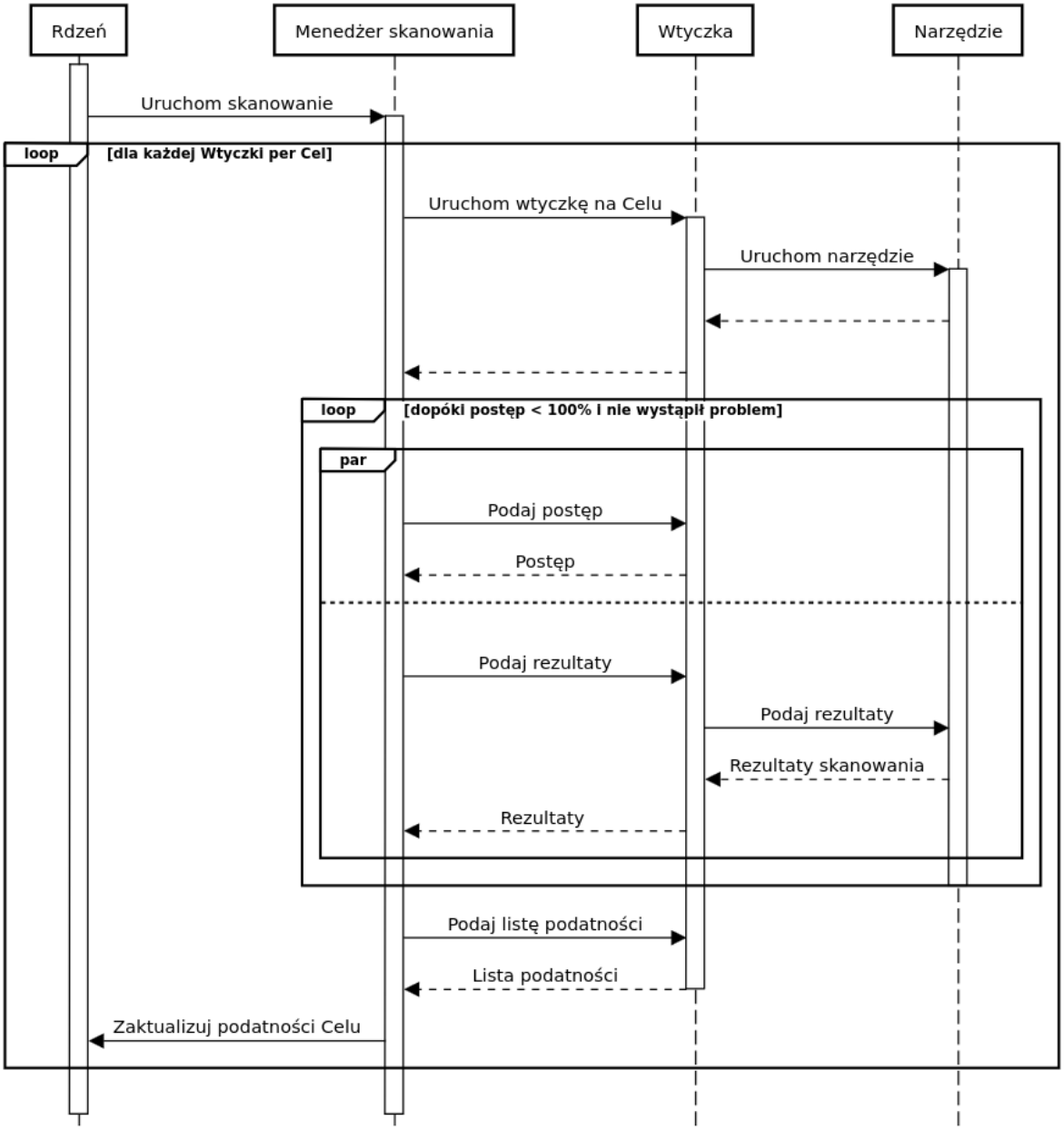
1. Ustalanie kolejności uruchamiania narzędzi.
2. Zależność pomiędzy narzędziami (jedno narzędzie wymaga konkretnie innego).
3. Ustalenie, co się dzieje, jeśli pośrednie narzędzie wykonało się niepoprawnie.
4. Decyzja, czy parsowanie danych, które mają być wprowadzone do drugiego programu, ma być wykonane w narzędziu zwracającym wynik, czy przyjmującym go jako dane wejściowe.
5. Ustalenie wspólnego formatu danych, który byłby możliwie generyczny dla całej koncepcji.

Powyższe zmiany nie są proste do wprowadzenia, zatem zastosowano inne podejście - w przypadku potrzeby takiego wykorzystania narzędzi, należałoby je uruchamiać jako jedną wtyczkę. Takie rozwiązanie przerzuca problem na użytkownika, ale jednocześnie pomija wymienione wyżej rzeczy, wymagające wprowadzenia dużych zmian w rdzeniu oraz wtyczkach, które miałyby mieć możliwość współdziałania z innymi.

Co jednak, gdy zajdzie potrzeba wielokrotnego wykorzystania wyniku jednej wtyczki jako danych wejściowych drugiej? Taka operacja kosztowałaby bardzo dużo czasu i te same skany przeprowadzane byłyby zbyt wiele razy. W tym scenariuszu najlepszą opcją będzie napisanie wtyczki typu *discovery* i odpowiednia modyfikacja kodu w rdzeniu. Zmiany polegałyby na gromadzeniu danych w bazie danych, nie jako wyniki (znalezione podatności), tylko ogólne dane o *Celu*. Wtedy każda wtyczka typu *scan* mogłaby się do nich odnieść i je wykorzystać. Przykładowe dane, które można gromadzić w taki sposób:

- lista domen/subdomen na danym hoście
- znalezione punkty końcowe API
- znalezione pliki na hoście
- znalezione adresy e-mail we wskazanej domenie

Podsumowując, w takim rozwiązaniu, wtyczki *discovery* mają pełnić rolę gromadzenia danych przed uruchomieniem wtyczek *scan*, co pozwoli na zaoszczędzenie czasu i zasobów. Z kolei wtyczki *scan* mogą uruchamiać więcej niż jedno narzędzie, w odpowiedniej kolejności,



Rysunek 3.1: Diagram sekwencyjny głównych zadań koncepcji

uwzględniając ewentualną konieczność przesyłania informacji między nimi. Dodatkowo, mogą wykorzystywać ustandaryzowane dane zgromadzone wcześniej przez wtyczki *discovery* pobierając je z bazy.

3.5.2 Używanie tylko wtyczek zgodnych z usługą

Gdyby każda wtyczka była uruchamiana na każdy działający serwis, to każda wtyczka musiałaby wstępnie weryfikować, czy usługa jest zgodna z tym, co wtyczka robi. Zajmowałoby to wiele czasu i wprowadzałoby zamęt w implementacji każdej wtyczki. Zamiast tego prostszym rozwiązaniem jest identyfikacja serwisu w czasie przeprowadzania *discovery*. Choć ilość serwisów nie jest ograniczona (każdy może stworzyć swój własny serwis z własnym protokołem), to - jak zostało pokazane w poprzednim rozdziale pracy - w skanowaniu chodzi w dużej mierze o testowanie powtarzalnych schematów. Z tego powodu można ograniczyć się do skończonej listy znanych usług. Taką listę stworzyli twórcy programu *nmap*, który również służy do skanowania otwartych portów. Wspomniana lista dostępna jest pod linkiem [50]. Serwis jest więc oznaczany nazwą z pierwszej kolumny. Te same nazwy serwisów muszą zostać użyte we wtyczce do oznaczenia tych obsługiwanych. Na tej podstawie rdzeń będzie uruchamiał tylko wtyczki, które obsłużą usługę na danym porcie.

3.5.3 Unikanie duplikatów podatności

Uruchamiając kolejne skany na tych samych *Celach* niemal pewna jest sytuacja, że podatności będą się duplikowały. O ile najistotniejsze podatności powinny być szybko naprawiane, to te o najmniejszym bądź informacyjnym ryzyku mogą nie być istotne. Aby uniknąć duplikatów, wtyczki muszą zwracać podatności w tej samej formie. Przykładowo, *Otwarta usługa SSH* jest tytułem ogólnym i jeśli wystąpiłby dwa razy, można by to zauważyć. Trzeba także uwzględnić inne dane, takie jak skanowany *Cel* i port, bo w przeciwnym razie w całym projekcie mogłaby wystąpić tylko jedna podatność *Otwarta usługa SSH*. Prostym rozwiązaniem na ten problem jest tworzenie sumy kontrolnej z wymienionych właściwości. Należy zauważyć, że ryzyko podatności nie powinno być używane w tym porównaniu (czy też wyliczaniu sumy kontrolnej), bo może zostać ręcznie zmienione. Taka sytuacja będzie najczęściej występowała przy oznaczaniu podatności jako fałszywy alarm (*False Positive*). W przeciwnym razie taka podatność pojawiałaby się w liście aktywnych podatności po każdym skanie, a do tego miałaby różne ryzyka.

Niestety takie rozwiązanie problemu ma też swoje wady - jeśli oznaczy się coś jako *False Positive*, to w przyszłości nie ma już możliwości na wykrycie takiej podatności na tym konkretnym *Celu* i porcie.

3.5.4 Wykrywanie naprawy podatności

Jeśli po wykryciu podatności zostanie ona załatana w aplikacji, nie powinna być ona już wykrywana przez wtyczki. Problemem jest jednak to, że podatność została już przypisana do danej usługi. Żeby automatycznie usunąć taką podatność, należałoby usuwać listę znalezionych przez wtyczkę podatności przed dodaniem świeżych rezultatów. Takie rozwiązanie będzie skuteczne w tym konkretnym problemie, natomiast tworzy problemy w innych aspektach:

- Podatności nie zawsze są wykrywane za każdym razem. Najprostszym przypadkiem będzie chwilowy problem połączenia się z aplikacją - wtyczka uzna, że serwer nie zwrócił odpowiedniego rezultatu dla danej reguły i będzie skanował dalej.

- Podatności znalezione wcześniej nie mogłyby być po prostu usunięte ze względu na zapisane w nich informacje. Przykładowo, użytkownik skanera mógłby już zmodyfikować ryzyko podatności, więc jej usunięcie i dodanie na nowo spowodowałoby, że ryzyko znów będzie takie, jakie ustawiła wtyczka. Aby temu zapobiec, należałoby zestawiać wcześniejsze podatności z obecnymi - pokrywające się pozostawiać bez zmian (tak jak obecnie są wpisane w aplikacji), a te, które były znalezione wcześniej, ale nie pojawiały się w nowym skanie - usuwać.

Ze względu na pierwszy aspekt wydaje się, że najlepszym rozwiązaniem jest ręczne usuwanie podatności przez użytkownika. Być może implementacja różnych opcji rozwiązałaby ten dylemat - wtedy użytkownik mógłby samodzielnie zadecydować, czy potencjalne ryzyko usunięcia niezałatanej podatności z listy jest istotne.

3.5.5 Skanowanie tylko wybranych portów

Nie w każdym scenariuszu użytkownik chce skanować wszystkie usługi działające na hoście. W przypadku chęci wprowadzenia takiego ograniczenia należałoby skanować jedynie wybrane porty. Najlepszym rozwiązaniem byłoby podawanie zakresów w formie *od-do* oraz po przecinkach, czyli łącznie np. `53,80,443,1000-5000`. Jeszcze jednym udogodnieniem może być ustalenie, czy zakres ma być białą czy czarną listą (*whitelist/blacklist*). Jeśli użytkownik chciałby omijać jedną konkretną usługę (port) to wygodniejszym mechanizmem byłaby lista czarna.

3.5.6 Ograniczenie wykorzystania łącza

Są różne metody na ograniczanie zużycia łącza:

- **Kontrola przepustowości na poziomie systemu operacyjnego**: Wiele systemów operacyjnych oferuje mechanizmy do kontroli przepustowości sieciowej. W systemie Linux można skorzystać z narzędzi takich jak *tc* (Traffic Control), które pozwalają na konfigurację ograniczeń na poziomie interfejsu sieciowego.
- **Ograniczenia na poziomie kontenera**: Jeśli aplikacje są uruchamiane w kontenerach takich jak *Docker*, można skonfigurować ograniczenia sieci na poziomie kontenera. *Docker* umożliwia ustawianie limitów przepustowości sieciowej podczas uruchamiania kontenera.

3.5.7 Uruchamianie wtyczki na wielu hostach/portach jednocześnie

Niektóre wtyczki mogą chcieć obsługiwać wiele portów jednocześnie. Przykładem może być narzędzie *nmap*, które da się skonfigurować w taki sposób, że skanowane jest wiele portów na raz. Wykorzystanie możliwości narzędzi w ten sposób może potencjalnie zaoszczędzić czas. Takie rozwiązanie prowadzi jednak do większej zawiłości implementacyjnej. Po pierwsze, istniałaby konieczność wprowadzenia innego sposobu zlecania prac wtyczkom, które są w stanie skanować kilka portów jednocześnie. Drugim problemem jest stabilność rozwiązania - przy uruchomieniu narzędzia 10 razy z małymi zadaniami będzie mniejsza szansa na np. jego zacięcie się, niż przy uruchomieniu narzędzia raz z dużą listą portów do skanowania. Dodatkowo w przyszłości może pojawić się problem przy implementacji limitu czasu działania pojedynczego narzędzia. Najlepiej jest zatem założyć, że jedno uruchomienie wtyczki może przeskanować jeden port (jedną usługę).

3.5.8 Wielowątkowość

Wstępna koncepcja nie zakłada wprowadzania obsługi wielowątkowości w rozwiązaniu. Poniżej znajdują się zagadnienia, z którymi należałoby się zmierzyć, aby wykorzystać wielowątkowość:

- Aby skaner był skuteczny musi wykonywać dużo zapytań. Ich redukcja jest możliwa, ale ograniczona. Jednocześnie liczba znanych podatności (m.in. tych, które mają przyznane numery CVE) rośnie, a aplikacje powiększają się. Wprowadzenie wielowątkowości - a tym samym zwiększenie ruchu ze strony skanujących wtyczek - podnosi ryzyko dla aplikacji oraz jej infrastruktury, ponieważ będą musiały zmagać się z dużą liczbą przychodzących zapytań. W konsekwencji może to doprowadzić do Odmowy dostępu do usługi (*Denial of Service*). Duży ruch sieciowy może także wywołać systemy alarmowe działające na testowanych hostach - w konsekwencji skanowanie będzie zatrzymane, a interwencja ze strony administratorów stworzy dodatkowe, niepotrzebne problemy.
- Uruchamianie wielu skanerów na raz może powodować duże wykorzystanie różnych zasobów na hoście. W skrajnych przypadkach może dochodzić do sytuacji, że skanery wzajemnie wyczerpią sobie zasoby i nie będą mogły wykonać swoich zadań.
- Wielowątkowość może zostać zastąpiona szerszym rozwiązaniem - przeniesieniem części działań na zewnętrzne środowiska np. poprzez uruchamianie zadań w chmurze lub na zewnętrznych serwerach. Szerszy opis tego tematu znajduje się w sekcji Skalowalność.
- Założeniem rozwiązania jest uruchamianie różnych zewnętrznych narzędzi. Problemem jest to, że każde z narzędzi będzie się lekko różniło konfiguracją ilości wątków, zatem ich globalne ustawienie z poziomu rdzenia może stanowić problem zarówno implementacyjny, jak i użytkowy dla końcowego użytkownika. Niektóre narzędzia mogą także nie obsługiwać zarządzania wątkami, więc globalne ograniczanie (np. dla konkretnego hosta) nie byłoby w pełni respektowane.
- Samo wprowadzenie wielowątkowości może istotnie zwiększyć poziom złożoności aplikacji.
- Zakładając użycie wielowątkowości, dobór odpowiedniego środowiska uruchomieniowego i języka programowania może mieć duże znaczenie, aby implementacja była prosta, a obsługa wielowątkowości jak najskuteczniejsza [51].

3.5.9 Skalowalność

Utrzymanie środowiska ze wszystkimi narzędziami może być problematyczne z wielu powodów:

- konieczność instalacji wszystkich narzędzi w jednym miejscu,
- duże wahania w wykorzystaniu zasobów w zależności od używanego narzędzia,
- długi i trudny proces instalacji (przede wszystkim narzędzi do wtyczek),
- rosnąca liniowo względem obu parametrów (ilości hostów i ilości wtyczek) złożoność skanowań wraz z rosnącą liczbą hostów i wtyczek: `O(liczba hostów * liczba wtyczek)`.

Zakładając chęć stworzenia rozwiązania działającego na pojedynczym hoście, dobrym rozwiązaniem wydaje się konteneryzacja. Użytkownik mógłby pobrać gotowy obraz, który po uruchomieniu miałby zainstalowane i skonfigurowane wszystkie narzędzia wymagane przez wtyczki.

Powyższe nie rozwiązuje jednak teoretycznie możliwego problemu wykorzystywania zasobów oraz rosnącego czasu skanowania wraz ze wzrostem liczby skanowanych hostów oraz wtyczek. Waga problemu mogłaby być zmniejszona przez wielowątkowość, ale niesie to za sobą dodatkowe problemy opisane w podsekcji Wielowątkowość. W przypadku skanowania różnych hostów, przy odpowiedniej implementacji, nie występowałby jednak problem generowania zbyt dużego ruchu do jednego hosta. Można to osiągnąć np. poprzez danie możliwości działania tylko jednej wtyczce na jednym hoście jednocześnie, ale w tym samym czasie uruchamiając tę lub inną wtyczkę na innych hostach. Generowany ruch wyjściowy z hosta, na którym działałoby rozwiązanie byłby duży, ale rozdzielałby się na testowane hosty.

Nie jest to jednak rozwiązanie na wszystkie problemy, dlatego należy także rozważyć uruchamianie narzędzi *w chmurze*, na żądanie. Skaner prawdopodobnie nie pracowałby non stop, więc wygodna byłaby możliwość uruchomienia narzędzi skanujących tylko na chwilę. Zakładając, że instancja wtyczki działałaby jako wirtualny serwer, możliwe byłoby uruchomienie dużej ilości skanerów jednocześnie. W takim rozwiązaniu rdzeń koncepcji pełniłby jedynie role zarządzania kolejnością wykonywania działań, przesyłania informacji, gromadzenia danych o podatnościach oraz interfejsu dla użytkownika. Ta metoda podejścia zwiększa jednak stopień trudności instalacji rozwiązania dla przeciętnego użytkownika, ale stanowi fundament do użycia go jako *Oprogramowanie jako usługa (Software as a Service, SaaS)*.

3.5.9.1 Aktualizacje narzędzi

Zakładając używanie zewnętrznych narzędzi, należy wziąć pod uwagę ich aktualizację. Niektóre z nich mogą poprawiać stabilność, inne dodawać funkcje, a jeszcze inne rozwijać bazę podatności. Najmniej istotne z perspektywy użytkownika będzie dodawanie nowych funkcji, ponieważ najprawdopodobniej będzie on korzystał z domyślnej konfiguracji narzędzia ustawionej w kodzie rozwiązania. Tym samym użytkownik nie zauważy dodania nowej funkcji. Poprawki dotyczące stabilności oraz takie, które poprawiają skuteczność skanów (również poprzez dodanie nowych podatności), są jednak bardzo istotne, ponieważ mogą bezpośrednio wpłynąć na rezultaty skanowania - potencjalnie mogą zostać wykryte inne podatności, może wystąpić mniej fałszywych alarmów oraz skan może wykonać się szybciej.

Powyższe dowodzi, że koncepcja musi uwzględniać zagadnienie aktualizacji używanych przez nią narzędzi. W większości przypadków używane narzędzia będą otwartoźródłowe, zatem dla uproszczenia założono, że ich instalacja będzie polegała na pobieraniu aplikacji np. z *GitHuba*. Wątek aktualizacji trzeba także połączyć z wybraną metodą skalowalności, więc nie będzie rozwinięty w każdym kierunku. Propozycje znajdują się poniżej:

1. **Aktualizowanie aplikacji przy budowaniu kontenera/kontenerów**

   Przy budowaniu kontenera, np. przy użyciu *Docker*, można zdecydować się na pobieranie zawsze najnowszych wersji aplikacji (np. najnowsze wydanie z *GitHuba*). W przypadku posiadania już działającej instancji, użytkownik mógłby ponownie zbudować kontener z koncepcją, jednocześnie zachowując dane dotyczące dodanych hostów czy też wykonanych skanów, ponieważ znajdowałyby się w oddzielnym kontenerze związanym z bazą danych lub oddzielnym *volume*.

   Takie rozwiązanie niesie za sobą ryzyko, że pobrane narzędzia przestaną działać. Przykładowo, narzędzie może zmienić sposób uruchamiania lub twórca może popełnić błąd, którego nie zauważy. Sposobem na przywrócenie aplikacji do działania bez ręcznego pobierania starszej wersji narzędzia jest powrót do starego kontenera. W

tym przypadku musimy pamiętać, żeby nie usuwać starego kontenera z koncepcją i narzędziami przed przetestowaniem czy wtyczki działają poprawnie.

2. **Tworzenie konfiguracji kontenera przez twórcę koncepcji i wsparcie społeczności**

   Te podejście zakłada, że kontener będzie podczas budowy pobierał konkretne wersje narzędzi. Te wersje będzie ustalać twórca koncepcji oraz społeczność na podstawie własnych testów. Wtedy prawdopodobieństwo, że narzędzie przestanie działać jest mniejsze, ale jednocześnie szybkość rozwoju koncepcji będzie wolniejsza i wymagająca ręcznej pracy.

Żadna z powyższych koncepcji nie jest idealna, a największym problemem jest prawdopodobnie wybór między stabilnością rozwiązania a szybkością rozwoju - po drodze mieszając konieczność ręcznej weryfikacji narzędzi. Być może dobrym podejściem będzie coś po środku - automatyczna aktualizacja niektórych narzędzi, a w przypadku innych jedynie ręczna.

3.5.9.2 Zmiana konfiguracji narzędzi

Założenie, że narzędzie w domyślnej konfiguracji będzie działało w każdym przypadku, jest błędne, ponieważ każda aplikacja oraz jej środowisko jest inne. Najprostszym przykładem może być wymóg autoryzacji np. poprzez przesyłanie odpowiedniego ciasteczka, aby dostać się do aplikacji. Takie problemy wymuszają wdrożenie zarządzania argumentami przekazywanymi do narzędzi uruchamianych przez wtyczki. Za najoptymalniejsze rozwiązanie wybrano ustawianie argumentów do wtyczek dla pojedynczych *Celów*. Rozwiązanie zbyt szerokie, tj. aktualizacja argumentów dla całej *Konfiguracji skanu* lub *Projektu*, mogłoby się nie sprawdzić np. w sytuacji, gdy tylko jeden host w projekcie wymaga wprowadzenia zmiany. Z kolei zmiana argumentu dla jednego *Celu* w całym projekcie mogłaby się nie sprawdzić w sytuacji, gdy użytkownik miałby np. dwie *Konfiguracje skanów* - mniej i bardziej intensywne. Konfiguracje narzędzi mogą być zatem inne dla każdej z nich. Zastosowano więc podejście, że zmianę w argumentach przekazywanych do narzędzia tworzy się dla konkretnego hosta w konkretnej konfiguracji skanu.

3.6 Przewaga przedstawionego rozwiązania nad konkurencją

Główną rzeczą, na którą trzeba patrzeć porównując narzędzia do skanowania, jest ich skuteczność, czyli ilość znalezionych podatności wraz ze zminimalizowaniem liczby fałszywych alarmów. W sekcji Porównanie narzędzi na podstawie rezultatów z poprzedniego rozdziału pokazane zostało, że różne narzędzia dają skrajnie różne rezultaty. Przedstawiona w tej pracy koncepcja ma rozwiązać ten problem, ponieważ daje możliwość integrowania wielu narzędzi w jedno większe. Skuteczność skanowania jest zatem, w uproszczeniu, sumą skuteczności narzędzi, które zostały dopisane jako wtyczki.

Otwarcie rozwiązania na modyfikacje ze strony innych użytkowników daje dodatkową przewagę nad rozwiązaniami komercyjnymi, które mogą nie adresować niektórych problemów. W przypadku komercyjnych rozwiązań, dodanie nowych funkcjonalności może być bardziej kwestią finansową (kalkulacja, czy dana funkcja jest warta poświęcenia czasu twórców), zaś w przypadku otwartego kodu może się okazać, że użytkownicy sami dopiszą upragnione funkcje i podzielą się tym z innymi. Należy również podkreślić fakt, że zewnętrzne narzędzia dopisywane do rozwiązania w formie wtyczek są także w wielu przypadkach otwartoźródłowe, co powiększa wspomniany efekt.

Cechą wyróżniającą rozwiązania na tle narzędzi komercyjnych będzie także to, że będzie darmowe. Może to dać mniejszym podmiotom możliwość monitorowania swoich zasobów bez opłacania kosztownych abonamentów, przy jednoczesnym satysfakcjonującym poziomie wykrywalności zagrożeń.

4. Opis narzędzi oraz technologii zastosowanych w pracy

Ta sekcja opisuje główne technologie wykorzystywane do stworzenia niniejszej pracy. Na początku opisane są technologie użyte bezpośrednio w implementacji. Następnie scharakteryzowano narzędzia, które ułatwiały proces powstawania implementacji oraz części pisemnej.

4.1 Python

Python to język programowania wysokiego poziomu, znany ze swojej czytelności i zwięzłej składni. Został stworzony przez Guido van Rossuma i po raz pierwszy opublikowany w 1991 roku. Jest on językiem interpretowanym, co oznacza, że nie jest wymagana kompilacja przed uruchomieniem programu [52].

Python występuje w dwóch głównych wersjach: *Python 2* oraz *Python 3*. Obie wersje posiadają swoje podwersje, np. 2.7 czy 3.9. Od stycznia 2020 *Python 2* stracił oficjalne wsparcie i nie jest dalej rozwijany. Trzeba także zauważyć, że *Python 2* nie jest kompatybilny z *Python 3*, zatem użytkownicy muszą podejmować decyzję, czy korzystać ze starej wersji (lub obu), czy może znaleźć alternatywne programy/biblioteki, które działają z nowszą wersją języka [53].[54]

Python charakteryzuje się wieloma wyróżniającymi się cechami:

- **Czytelność i prosta składnia**: *Python* ma składnię, która jest zrozumiała i łatwa do nauki, co czyni go dobrym językiem dla początkujących programistów. Składnia jest tak zaprojektowana, aby pomóc programistom skupić się na logice i funkcjonalności programu, a nie na skomplikowanych detalach składniowych.
- **Dynamiczne typowanie**: *Python* jest językiem dynamicznie typowanym, co oznacza, że nie trzeba deklarować typu zmiennych podczas ich tworzenia. *Python* sam zrozumie, jaki typ danych przypisać zmiennej na podstawie wartości, która została jej przypisana.
- **Wsparcie dla różnych paradygmatów programowania**: *Python* obsługuje różne paradygmaty programowania, takie jak programowanie obiektowe, proceduralne i funkcyjne. Dzięki temu programiści mogą wybrać styl, który najlepiej pasuje do ich problemu.
- **Bogata biblioteka standardowa**: *Python* ma obszerną bibliotekę standardową, która obejmuje wiele modułów pomocniczych, np. do operacji matematycznych, manipulacji plikami, komunikacji sieciowej etc.
- **Społeczność i ekosystem**: *Python* ma jedną z najaktywniejszych społeczności programistów na świecie - według [55] jedynie *JavaScript* posiada większą społeczność. Przekłada się to na mnogość dostępnych narzędzi, bibliotek i dokumentacji. To również sprawia, że *Python* jest niezwykle zasobnym językiem, który można wykorzystać do realizacji praktycznie dowolnego projektu.

Język *Python* ma wiele zastosowań:

- **Aplikacje internetowe**: Z użyciem Pythona można stworzyć aplikację internetową. Najbardziej popularne *frameworki*, które do tego służą, to *Django* i *Flask* [56].
- **Data science**: *Python* jest jednym z najpopularniejszych języków w dziedzinie nauki o danych, dzięki bibliotekom takim jak *pandas*, *NumPy* i *matplotlib*, które ułatwiają manipulację danymi i ich wizualizację[57].
- **Uczenie maszynowe i sztuczna inteligencja**: *Python* jest często wybierany jako język dla projektów związanych z uczeniem maszynowym i sztuczną inteligencją. Biblioteki

takie jak *TensorFlow*, *PyTorch* i *Scikit-learn* umożliwiają tworzenie skomplikowanych modeli uczenia maszynowego [58].

- **Automatyzacja i skryptowanie**: *Python* jest również popularny w automatyzacji zadań i skryptowaniu, dzięki łatwości pisania i uruchamiania skryptów, a także szerokiemu wsparciu dla różnych platform i systemów operacyjnych [53].

W 2017 opublikowano artykuł pt. *Python - Najszybciej rozwijający się język programowania* (*Python – The Fastest Growing Programming Language*)[59]. Wymieniono w nim kilka ciekawych informacji o popularności Pythona:

- *Python* był piąty w wielkości społeczności na *StackOverflow*
- *Python* był 3 największą społecznością na *Meetup.com* jeśli chodzi o języki programowania
- *Python* był czwartym najczęściej używanym językiem na *GitHub*

Python, mimo swojej popularności i wszechstronności, ma pewne wady, które mogą wpływać na wybór języka w określonych kontekstach. Przede wszystkim, *Python* nie jest tak szybki jak niektóre inne języki, takie jak *C++* lub *Java*. Jego interpretowana natura sprawia, że wykonanie programu może trwać dłużej, co może stanowić problem dla aplikacji wymagających dużej mocy obliczeniowej, takich jak gry wideo czy programy inżynierskie.

Drugim problemem jest zarządzanie pamięcią. *Python* automatycznie zarządza pamięcią za pomocą mechanizmu *garbage collector*, który jest wygodny, ale nie zawsze najbardziej efektywny pod względem wykorzystania pamięci. Programiści, którzy potrzebują pełnej kontroli nad pamięcią, mogą odczuć to jako ograniczenie.

Trzeci aspekt dotyczy wykorzystania wielowątkowości. *Python* ma wewnętrzne ograniczenie związane z *Global Interpreter Lock (GIL)*, co oznacza, że tylko jeden wątek może być wykonywany na raz. To czyni Pythona mniej efektywnym w przypadku aplikacji, które muszą korzystać z wielowątkowości na wielordzeniowych procesorach. Jednym z rozwiązań jest użycie biblioteki **multiprocessing**, która umożliwia tworzenie osobnych procesów dla każdego zadania. Każdy z nich ma własną instancję interpretera Pythona i pamięć, co oznacza, że mogą działać równolegle na wielu rdzeniach procesora. Ta technika jest szczególnie efektywna dla zadań intensywnych pod względem obliczeń oraz tych, które można łatwo podzielić na niezależne podzadania [60].

4.2 Django

Django to framework do tworzenia aplikacji internetowych napisany w Pythonie. Nazwa pochodzi od imienia słynnego gitarzysty jazzowego *Django* Reinhardta [61], [62]. *Django* zostało zaprojektowane, aby pomóc programistom tworzyć złożone, bazodanowe aplikacje internetowe szybko i z minimalnym wysiłkiem. Oto kilka kluczowych cech Django:

- **Model-View-Template (MVT)**: *Django* używa architektury *MVT*, która jest podobna do *Model-View-Controller (MVC)*. *Model* reprezentuje dane, *View* jest odpowiedzialny za logikę biznesową, a *Template* odpowiada za prezentację danych użytkownikowi [63].
- **Bezpieczeństwo**: *Django* pomaga tworzyć bezpieczne aplikacje internetowe, domyślnie zapewniając ochronę przed typowymi atakami, takimi jak Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF), czy SQL Injection [64].
- **Object-relational mapping (ORM)**: *Django* ma wbudowany *ORM*, który ułatwia interakcję z bazą danych. Dzięki *ORM*, programiści mogą pracować z bazami danych na wysokim poziomie abstrakcji, używając instrukcji Pythona zamiast pisząc

zapytania SQL. *Django* obsługuje wiele systemów zarządzania bazami danych, takich jak *PostgreSQL*, *MySQL*, *SQLite* i *Oracle* [56].

Django może być szczególnie przydatny dla programistów chcących skupić się na tworzeniu funkcji aplikacji, a nie na niskopoziomowych detalach, takich jak obsługa protokołów HTTP czy zapytań do bazy danych. *Django* zapewnia gotowe do użycia moduły dla wielu typowych zadań związanych z tworzeniem aplikacji internetowych, co może znacznie przyspieszyć proces tworzenia nowych aplikacji.

4.3 Docker

Docker to otwarte oprogramowanie zaprojektowane do automatyzacji procesu wdrażania aplikacji jako przenośne, samowystarczalne kontenery, które mogą działać na dowolnym systemie operacyjnym. Używając *Dockera* można tworzyć obrazy (*images*) stanowiące fundament aplikacji wraz z jej zależnościami, a następnie uruchamiać je jako kontenery. Dzięki temu eliminowane są typowe problemy związane z różnicami między środowiskami produkcyjnymi a deweloperskimi. Stworzone obrazy mogą być wykorzystywane wielokrotnie do utworzenia wielu kontenerów [65].

Kluczowym aspektem efektywnego zarządzania obrazami i kontenerami jest mechanizm warstw. W procesie tworzenia obrazu *Docker* każda instrukcja zawarta w *Dockerfile* generuje nową warstwę obrazu. Owe warstwy reprezentują zmiany wprowadzone do obrazu przez poszczególne instrukcje. Te warstwy są nakładane na siebie w kolejności zdefiniowanej w *Dockerfile*, składając się na finalny obraz. Każda kolejna warstwa jest jedynie rozszerzeniem poprzedniej, co oznacza, że zawiera jedynie te zmiany, które zostały wprowadzone w danej instrukcji. Taka struktura umożliwia *Dockerowi* efektywne zarządzanie i udostępnianie warstw między różnymi obrazami i kontenerami. Gdy kontener jest uruchamiany z obrazu, *Docker* tworzy tzw. warstwę pisania na szczycie wszystkich warstw tylko do odczytu, które składają się na obraz. Kontener wykorzystuje tę warstwę do przechowywania wszystkich zmian, które są wprowadzane w trakcie jego działania. Takie podejście ma kilka zalet. Po pierwsze, pozwala na efektywne wykorzystanie przestrzeni dyskowej, ponieważ zmiany są przechowywane tylko raz, nawet jeśli są używane w wielu obrazach lub kontenerach. Po drugie, umożliwia szybkie uruchamianie kontenerów, ponieważ tylko warstwy, które nie są już przechowywane na lokalnym systemie, muszą być pobrane. Po trzecie, umożliwia wprowadzanie zmian w działającym kontenerze bez wpływu na obraz, na którym był on oparty, co zwiększa bezpieczeństwo i elastyczność[66].

Każdy kontener działa niezależnie od innych, co oznacza, że każda aplikacja jest izolowana od innych aplikacji, nawet jeśli są one uruchomione na tym samym hoście *Dockera*. Izolacja kontenerów zwiększa bezpieczeństwo i pozwala na niezależne zarządzanie każdym z nich. Z drugiej strony, przy odpowiedniej konfiguracji, możliwe jest konfigurowanie łączności między kontenerami np. poprzez reguły umożliwiające wzajemną komunikację sieciową. Takie rozwiązanie pozwala na wykorzystanie dwóch różnych kontenerów dla jednej aplikacji - przykładowo baza danych i serwer backendowy.

Docker zawiera także mechanizm wolumenów (*volumes*). Umożliwia on przechowywanie danych generowanych i używanych przez kontenery. Wolumeny stanowią dedykowaną część systemu plików, która jest zarządzana przez *Docker* i może być przypisana do jednego lub więcej kontenerów. Są niezależne od cyklu życia kontenera, co oznacza, że dane przechowywane na woluminach przetrwają nawet po usunięciu kontenera. Mechanizm wolumenów jest często wykorzystywany w sytuacjach, gdy użytkownik chce zachować dane między uruchomieniami kontenera, dzielić dane między wieloma kontenerami lub kiedy

potrzebne jest bezpośrednie mapowanie do hosta. W praktyce, woluminy są często używane do przechowywania bazy danych, logów aplikacji, czy też innych informacji, które mają być utrzymywane na dłuższy okres czasu.

Docker jest popularny w środowiskach DevOps ze względu na możliwości szybkiego wdrażania, skalowania oraz prostego zarządzania. Jest często używany do wdrażania mikroserwisów w chmurze, testowania oprogramowania w izolowanym środowisku, a także automatyzacji budowy i wdrażania oprogramowania [67].

4.4 PostgreSQL

PostgreSQL to zaawansowany system zarządzania bazami danych typu open source. Wspiera zarówno dane SQL (relacyjne), jak i JSON (nierelacyjne). Oferuje również szereg zaawansowanych funkcji, takich jak widoki, podzapytania, funkcje wbudowane i zewnętrzne oraz procedury składowane [68]. Podstawowe cechy *PostgreSQL* to:

- **ACID**: *PostgreSQL* jest w pełni zgodny z *ACID (Atomicity, Consistency, Isolation, Durability)*.
- **Liczne typy danych**: Obsługuje szeroki zakres typów danych, takich jak liczby całkowite, zmiennoprzecinkowe, dane tekstowe, daty i czasu, a także typy geometryczne i sieciowe.
- **Rozszerzenia**: *PostgreSQL* umożliwia tworzenie i korzystanie z rozszerzeń, które mogą dostarczać dodatkowe funkcje, takie jak indeksy *GIS*, typy danych *JSON*, *hstore* dla par klucz-wartość itp.
- **Bezpieczeństwo**: *PostgreSQL* oferuje silne mechanizmy bezpieczeństwa, takie jak uwierzytelnienie, szyfrowanie, role i uprawnienia (na poziomie bazy danych, a nie aplikacji).

Dla wielu aplikacji internetowych, *PostgreSQL* jest często preferowanym wyborem jako system zarządzania bazami danych. Jest to związane z jego niezawodnością, bezpieczeństwem, możliwościami skalowania i wszechstronnością. Znane *frameworki* aplikacji internetowych takie jak *Django* (*Python*) i *Ruby on Rails* (*Ruby*) mają dobre wsparcie dla *PostgreSQL*, co ułatwia jego implementację i wykorzystanie.

Porównanie *PostgreSQL* z MySQL pomaga zrozumieć, jakie są unikalne cechy i przewagi każdego z tych systemów zarządzania bazami danych.

PostgreSQL to obiektowo-relacyjny system zarządzania bazami danych, który jest uznawany za bardziej zaawansowany i elastyczny w porównaniu do *MySQL*. Obsługuje zarówno standard SQL, jak i rozszerzenia do SQL, w tym obsługę dla JSON i pełnotekstowego wyszukiwania. Ponadto, *PostgreSQL* oferuje transakcyjne DDL *(Data Definition Language)*, co oznacza, że nawet operacje takie jak tworzenie i modyfikowanie tabel są transakcyjne, co jest cechą nieosiągalną dla *MySQL*. Z drugiej strony, *MySQL*, będący relacyjnym systemem zarządzania bazami danych, jest zdecydowanie łatwiejszy w użyciu i ma prostszy interfejs. Jest on również znany z szybkości i efektywności, szczególnie w aplikacjach, które nie wymagają skomplikowanych transakcji lub specjalnych typów danych. *MySQL* jest również bardzo popularny wśród użytkowników początkujących ze względu na jego prostotę i szerokie wsparcie społeczności [69].

Obie te bazy danych są dobrze wspierane przez większość języków programowania i są często używane w rozwoju aplikacji webowych. Wybór między *PostgreSQL* a *MySQL* zależy od specyficznych wymagań projektu, w tym od złożoności danych, potrzeb transakcyjnych i przewidywanej skalowalności. Bazując na *StackOverflow Developer Survey*, *MySQL* i *PostgreSQL* stanowią razem ponad 80% rynku, a same są podobnie popularne [70].

4.5 PyCharm

PyCharm to zintegrowane środowisko programistyczne (Integrated Development Environment, IDE) stworzone przez *JetBrains*. Jest dedykowane dla języka Python, choć obsługuje również wiele innych języków. *PyCharm* jest dostępny w dwóch wersjach: *Community Edition*, która jest darmowa, oraz *Professional Edition*, która jest płatna i oferuje dodatkowe funkcje. Oto kilka kluczowych cech *PyCharm*:

- **Inteligentne podpowiedzi**: *PyCharm* oferuje funkcje, takie jak składnia podświetlania, autouzupełniania, inspekcje kodu i zaawansowane narzędzia do nawigacji, które pomagają w efektywnym i poprawnym kodowaniu.
- **Refaktoryzacja kodu**: *PyCharm* zawiera zestaw narzędzi do refaktoryzacji kodu, które umożliwiają łatwe przeprojektowanie struktury kodu bez zmiany jego funkcjonalności.
- **Debugger**: *PyCharm* zawiera potężny debugger z zestawem funkcji, takich jak punkty przerwania, kroki, wyrażenia i ocena itp., które pomagają w diagnozowaniu i rozwiązywaniu problemów z kodem.
- **Wsparcie dla testów**: *PyCharm* oferuje wsparcie dla różnych frameworków testowych, takich jak *Unittest*, *Nose*, *Pytest* itp., co umożliwia łatwe tworzenie, uruchamianie i debugowanie testów jednostkowych.
- **Wsparcie dla systemów kontroli wersji**: *PyCharm* integruje się z popularnymi systemami kontroli wersji, takimi jak *Git*, *SVN*, *Mercurial* itp., umożliwiając łatwą kontrolę wersji i współpracę.
- **Wsparcie dla technologii webowych**: W wersji Professional *PyCharm* oferuje wsparcie dla wielu technologii webowych, takich jak *Django*, *Flask*, *Google App Engine*, *Pyramid*, *web2py*, *HTML/CSS*, *JavaScript*, *TypeScript*, *AngularJS*, *Node.js* itp.
- **Integracja z *Docker***: *PyCharm* umożliwia integrację z narzędziami takimi jak *Docker*, co pozwala na lepszą automatyzację i zarządzanie środowiskiem deweloperskim.

Głównymi alternatywami dla *PyCharm* są *Visual Studio*, *Visual Studio Code*, *Atom*, *Sublime Text* oraz *Eclipse* z wtyczką *PyDev*.

4.6 ChatGPT

ChatGPT to model języka naturalnego stworzony przez *OpenAI* na bazie architektury *GPT (Generative Pretrained Transformer)*. Jest on zdolny do generowania odpowiedzi na pytania, pisania esejów, tłumaczeń języków i wykonywania wielu innych zadań związanych z językiem naturalnym. Model *GPT*, na którym opiera się *ChatGPT*, jest modelem opartym na transformerach, który jest pre-trenowany na dużych korpusach tekstu i następnie dostrajany (fine-tuned) na specyficzne zadanie, takie jak generowanie odpowiedzi na pytania w formie dialogu [71]. Główne cechy *ChatGPT*:

- **Generowanie tekstu**: *ChatGPT* potrafi generować długie, spójne fragmenty tekstu, które są gramatycznie poprawne i są odpowiedzią na podane mu pytania lub polecenia.
- **Rozumienie kontekstu**: *ChatGPT* potrafi zrozumieć i odpowiedzieć na pytania w kontekście poprzednich wypowiedzi w konwersacji.
- **Wielojęzyczność**: *ChatGPT* potrafi generować teksty w wielu językach, choć jego podstawowy trening odbywał się na tekstach w języku angielskim.

W kontekście programowania, *ChatGPT* może być użytecznym narzędziem dla programistów. Oto kilka potencjalnych zastosowań:

- **Podpowiedzi i naprawa kodu**: *ChatGPT* może analizować kod i sugerować poprawki lub ulepszenia. Dzięki swojej zdolności do przetwarzania języka naturalnego może zrozumieć pytania programisty i udzielać odpowiedzi na podstawie wiedzy z treningowych danych. Może to obejmować podpowiedzi dotyczące składni, wydajności kodu lub poprawnej implementacji funkcji.
- **Refaktoryzacja kodu**: *ChatGPT* może pomagać w procesie refaktoryzacji, sugerując, gdzie kod może być uproszczony, ulepszony lub zreorganizowany. Jego zdolność do analizowania tekstu i generowania odpowiedzi na pytania może być użyteczna w identyfikowaniu obszarów kodu, które mogą być zrefaktoryzowane.
- **Automatyczne generowanie dokumentacji**: Dzięki zdolności generowania spójnego tekstu, *ChatGPT* może być użyteczny w tworzeniu dokumentacji kodu. Może generować opisy funkcji, klas, metod, komentarze do kodu i inne elementy dokumentacji na podstawie informacji zawartych w kodzie.

W toku tworzenia niniejszej pracy *ChatGPT* był używany głównie jako pomoc przy pisaniu kodu. Przykładowe zadania, na które *ChatGPT* potrafił znaleźć rozwiązanie:

- ***Pokaż kilka metod na utworzenie singletona w Python3. Następnie porównaj wady i zalety każdego rozwiązania.***

  ChatGPT odpowiedział na pytanie podając przykładowe rozwiązania i uargumentował, które jest najlepsze.
- ***Podaj funkcję w Django, która pozwala na zrobienie X.***

  Wielokrotnie *ChatGPT* potrafił podać funkcję wbudowaną we *framework Django*, której nie znał autor pracy, ale był w stanie opisać co miałaby robić. Podobny rezultat można osiągnąć poprzez poproszenie o podanie odpowiednika konkretnej funkcji z innego *frameworka*/języka programowania.
- ***Zoptymalizuj ten kod, aby z dwóch klas stworzyć jedną.***

  W momencie, w którym w kodzie widać było niepotrzebnie utworzoną oddzielną klasę, mającą podobne zadanie do innej, potrzebny był *refactoring* kodu. Eksperymentalnie wklejono kod dwóch klas w *ChatGPT*, który był w stanie połączyć je w jedną, uwzględniając istniejące między nimi różnice.

Technologia *ChatGPT* była zatem używana zamiennie z wyszukiwarkami internetowymi oraz dokumentacjami. Odpowiedzi narzędzia firmy *OpenAI* charakteryzowały się przystępną formą wyświetlonych odpowiedzi - odpowiadał konkretnie na zadane pytania i pokazywał przykłady. W przypadku wyszukiwarek internetowych, należałoby najpierw wpisać szukane hasło, a potem przejrzeć strony internetowe, które zawierają odpowiedź. Z drugiej strony, *ChatGPT* wymaga czasu na odpowiedź, a te z kolei mogą być niesatysfakcjonujące - zauważono, że narzędzie potrafi wymyślić odpowiedź, która nie ma większego sensu. Podobne problemy występowały z generowaniem kawałków kodu - odpowiedzi zawierały kod, który nie mógł działać np. dlatego, że mieszał funkcje z różnych bibliotek. *ChatGPT* wymagał również weryfikacji poprawności wskazanego rozwiązania: W przypadku stron internetowych można mówić o ich reputacji oraz można korzystać z mechanizmu oceniania odpowiedzi (np. na *StackOverflow*). W *ChatGPT* odpowiedzi były zwracane zawsze w podobny sposób, a źródło, na którym się wzorował, pozostawało nieznane.

Niniejsza praca pisemna została napisana w języku *LaTeX*. Aby dostosować szablon dokumentu proszono *ChatGPT* o podanie rozwiązań na niektóre problemy takie jak pozycjonowanie obrazków czy wskazywanie funkcji zmieniających określone elementy. Narzędzie pomagało również w zadaniach, takich jak przepisanie list na język *LaTeX*. Problem polegał na potrzebie dodania znaczników `\begin{itemize}` przed listą, `\end{itemize}` po liście

oraz `\item` przed każdym elementem listy. *ChatGPT* był w stanie wychwytywać niuanse, takie jak potrzeba dodania znaków ucieczki (*escaping*) przed niektórymi znakami, co także ułatwiało proces pisania tekstu. Takie operacje mogą zaoszczędzić czas, a jednocześnie dają konkretny rezultat - w przypadku generowania kodu może wystąpić zdecydowanie więcej błędów niż przy tak schematycznym przykładzie jak ten z listami.

Podsumowując, *ChatGPT* może ułatwić oraz przyśpieszyć proces tworzenia kodu oraz rozwiązywania błędów, które się w nim znajdują. Technologia ta nie jest jednak nieomylna, więc należy weryfikować, czy wygenerowany kod jest zgodny z oczekiwaniami oraz czy nie zawiera podatności bezpieczeństwa.

4.7 GitHub Copilot

GitHub Copilot to narzędzie asystujące programistom w pisaniu kodu, stworzone przez *GitHub* we współpracy z *OpenAI* [72]. Jest ono dostępne jako wtyczka do różnych IDE i służy jako *inteligentna* funkcja autouzupełniania kodu sugerująca fragmenty kodu podczas pisania.

GitHub Copilot generuje sugestie dla całych linii lub bloków kodu na podstawie kontekstu wpisywania. Działa to nie tylko dla funkcji i metod, ale również dla komentarzy, łańcuchów znaków i innych fragmentów kodu. *Copilot* został wytrenowany na różnych językach i technologiach, co oznacza, że może generować sugestie kodu dla wielu języków programowania, w tym *JavaScript*, *Python*, *TypeScript*, *Ruby*, *Java* i wielu innych. *GitHub Copilot* został wytrenowany na miliardach linii kodu z publicznych repozytoriów na *GitHub*, dzięki czemu ma dostęp do ogromnej bazy wiedzy dotyczącej różnych wzorców kodowania i rozwiązań problemów [72]. *GitHub Copilot* może być użyteczny w wielu aspektach pisania kodu:

- **Przyspieszenie pisania kodu**: Dzięki automatycznemu generowaniu sugestii, *Copilot* może przyspieszyć proces tworzenia kodu, pomagając programistom szybko napisać linie lub bloki kodu.
- **Uczenie się nowych technologii**: *Copilot* może pomóc programistom nauczyć się nowych języków i technologii, oferując sugestie kodu oparte na powszechnych wzorcach. *Copilot* może być przydatny w momentach, kiedy programista wie, jaki cel chce osiągnąć, ale nie zna dokładnej funkcji, która to robi w danym języku programowania.
- **Wspomaganie rozwiązywania problemów**: Dzięki swojej zdolności do generowania różnych rozwiązań na podstawie kontekstu, *Copilot* może pomóc programistom w generowaniu kodu do bardziej skomplikowanych zadań. *Copilot*, korzystając ze swojej *wiedzy*, jest w stanie wygenerować nawet wielolinijkowe rozwiązania i wykorzystać zmienne użyte wcześniej przez programistę.

W toku pisania niniejszej pracy *GitHub Copilot* był używany w formie wtyczki do programu *PyCharm*. Po wstępnej konfiguracji wtyczka podpowiadała składnię w podobny sposób jak wbudowane w *IDE* od *JetBrains* mechanizmy, ale często wychodziła dużo dalej. Przykładowo, *Copilot* był w stanie generować kod funkcji na podstawie jej nazwy i kodu, który już istniał w otwartym pliku. Czasem sugestie były na tyle dokładne, że nie wymagały wprowadzania żadnych zmian - to były zazwyczaj sytuacje, w których tworzono funkcje podobne do innych już obecnych. *Copilot* potrafił także realizować zadania podobnie jak *ChatGPT*: można było stworzyć funkcję i napisać komentarz, w którym zawarto informację co dana funkcja ma robić. Po chwili narzędzie było w stanie napisać przykładowy kod takiej funkcji. Zauważono, że sugestie *Copilota* były najskuteczniejsze w przypadku wykonywania szablonowych działań, np.:

- tworzenie funkcji do obsługi działań związanych z bazą danych (np. zapis, odczyt, przefiltrowanie rezultatów)
- tworzenie kodu w HTML oraz widoków w Jinja2
- tworzenie kodu podobnego do już napisanego

Podobnie jak w przypadku *ChatGPT*, *GitHub Copilot* nie jest doskonały i wymaga nadzoru człowieka. Wielokrotnie widoczne były podpowiedzi, które nie miały sensu bądź były zwyczajnie niepoprawne. Jego sugestie powinny być zatem zawsze sprawdzane pod kątem poprawności i bezpieczeństwa. *Copilot* ma mniejsze możliwości od *ChatGPT* (ogranicza się tylko do kodu), ale lepiej spełnia swoją funkcję ze względu na większą świadomość kontekstu (ma wgląd w pisany kod, a nie tylko fragmenty) oraz szybkość. W obecnej formie narzędzie może być bardzo pomocne dla programisty, ale nie ma możliwości, aby go zastąpił. Niektóre podpowiedzi tego narzędzia mogą zaoszczędzić czas potrzebny na przejrzenie dokumentacji bądź wyszukanie czegoś w przeglądarce.

4.8 Overleaf

Overleaf jest zaawansowanym narzędziem online umożliwiającym edycję i kolaborację w czasie rzeczywistym dla dokumentów stworzonych przy użyciu języka *LaTeX*. Narzędzie to przekształca proces tworzenia i edycji dokumentów *LaTeX*, umożliwiając użytkownikom ich tworzenie, modyfikowanie i udostępnianie bezpośrednio w przeglądarce internetowej - bez konieczności instalowania dodatkowego oprogramowania na lokalnym komputerze [73].

Jednym z kluczowych aspektów *Overleaf* jest możliwość współpracy w czasie rzeczywistym. Umożliwia ona wielu użytkownikom jednoczesną pracę nad tym samym dokumentem, co pozwala na obserwację i śledzenie wprowadzanych przez nich zmian na bieżąco. Jest to funkcja szczególnie cenna w kontekście pracy zespołowej, gdzie często występuje konieczność współpracy nad skomplikowanymi dokumentami.

Overleaf wyposażony jest w zintegrowane środowisko do pisania w *LaTeX*, które zawiera takie funkcje jak podświetlanie składni, automatyczne uzupełnianie kodu i inne, które ułatwiają pisanie dokumentów. Dodatkowym atutem platformy jest również szeroki wybór dostępnych szablonów dokumentów, które mogą służyć jako punkt wyjścia dla nowych projektów.

Narzędzie posiada funkcję zarządzania wersjami, która umożliwia śledzenie i przywracanie zmian, a także integrację z systemami kontroli wersji, takimi jak *GitHub*. Ponadto, *Overleaf* umożliwia kompilację i przeglądanie wynikowego dokumentu PDF bezpośrednio w przeglądarce, co może ułatwić i przyśpieszyć proces tworzenia dokumentów *LaTeX* - szczególnie dla użytkowników, którzy nie chcą instalować środowiska na lokalnym komputerze.

5. Techniczne aspekty pracy

Większość tego rozdziału jest poświęcona implementacji koncepcji opisanej w rozdziale Generyczny skaner bezpieczeństwa. Stworzone narzędzie nazwano *RedMirror*. Człon *Red* ma nawiązywać do pojęcia *Red Team*, które odnosi się do grupy specjalistów przeprowadzających kontrolowane ataki na systemy informatyczne. Celem *Red Teamu* jest odkrywanie luk w zabezpieczeniach, znajdowanie podatności systemu, testowanie skuteczności procedur reagowania na incydenty oraz sprawdzanie skuteczności i skutków działań zabezpieczających. Drugi człon - *Mirror (lustro)* - odnosi się to założeń koncepcji polegających na uruchamianiu tych samych konfiguracji dla różnych hostów.

Końcówka tego rozdziału zawiera także inne techniczne informacje dotyczące rzeczy wykorzystanych w toku tworzenia pracy, ale nie związanych bezpośrednio z koncepcją Generycznego skanera bezpieczeństwa.

5.1 Stworzone rozwiązanie

Efektem pracy nad implementacją koncepcji Generycznego skanera bezpieczeństwa jest narzędzie *RedMirror*, które pozwala na skanowanie zasobów sieciowych oraz zarządzanie tym procesem poprzez aplikację webową. Implementacja obejmuje główne założenia koncepcji i jest możliwa do używania. Obecny stan rozwoju można określić jako *alfa*, przez co można rozumieć możliwość korzystania z aplikacji i zaimplementowanych funkcji, jednocześnie wiedząc, że aplikacja wymaga dalszego rozwoju, poprawiania i testowania, w szczególności aby aplikacja była niezawodna.

5.1.1 Zaimplementowane funkcjonalności

- zarządzanie Projektami
- zarządzanie Celami
- tworzenie konfiguracji skanów
- uruchamianie skanów
- zapisywanie rezultatów skanów
- odczyt wyników skanów w formie tekstowej
- odczyt wyników skanów w formie podatności przypisanych do poszczególnych serwisów
- ustalanie ryzyka podatności

5.1.2 Ekrany aplikacji

Ta podsekcja składa się ze zrzutów ekranu z aplikacji oraz opisu funkcjonalności bądź danych, które są na nich widoczne.

Rysunek 5.1 pokazuje ekran z projektami, który jest domyślnym ekranem aplikacji, wyświetlającym listę stworzonych projektów. Można je dodawać, usuwać, edytować bądź otwierać.

Rysunek 5.2 przedstawia *dashboard* wskazanego projektu. Znajdują się na nim informacje o stworzonych *Celach*, Konfiguracjach skanów oraz informacje o ostatnich 5 uruchomionych skanach.

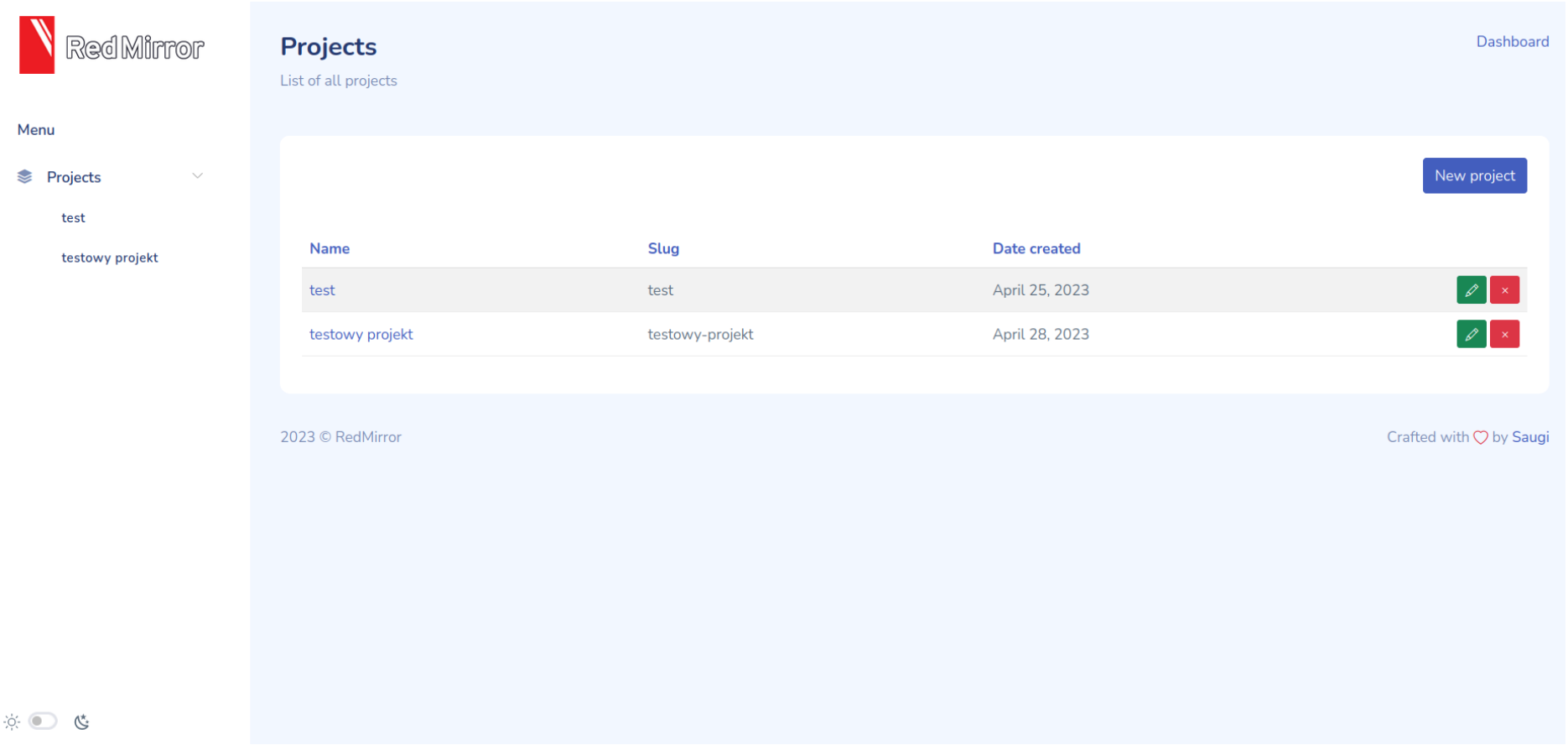
Rysunek 5.3 zawiera ekran edycji konfiguracji skanu. W konfiguracji skanu ustala się na jakich *Celach* ma być wykonywane skanowanie oraz jakie wtyczki mają być użyte. Można tu także nadpisać domyślne argumenty przekazywane do narzędzia.

Rysunek 5.4 pokazuje *dashboard* pojedynczego Celu, na którym pokazane są serwisy działające na hoście (otwarte porty). Dla każdego z nich pokazane są informacje, jaki to serwis, jego baner oraz liczbę podatności dla różnych ryzyk (każde ryzyko ma swój kolor).

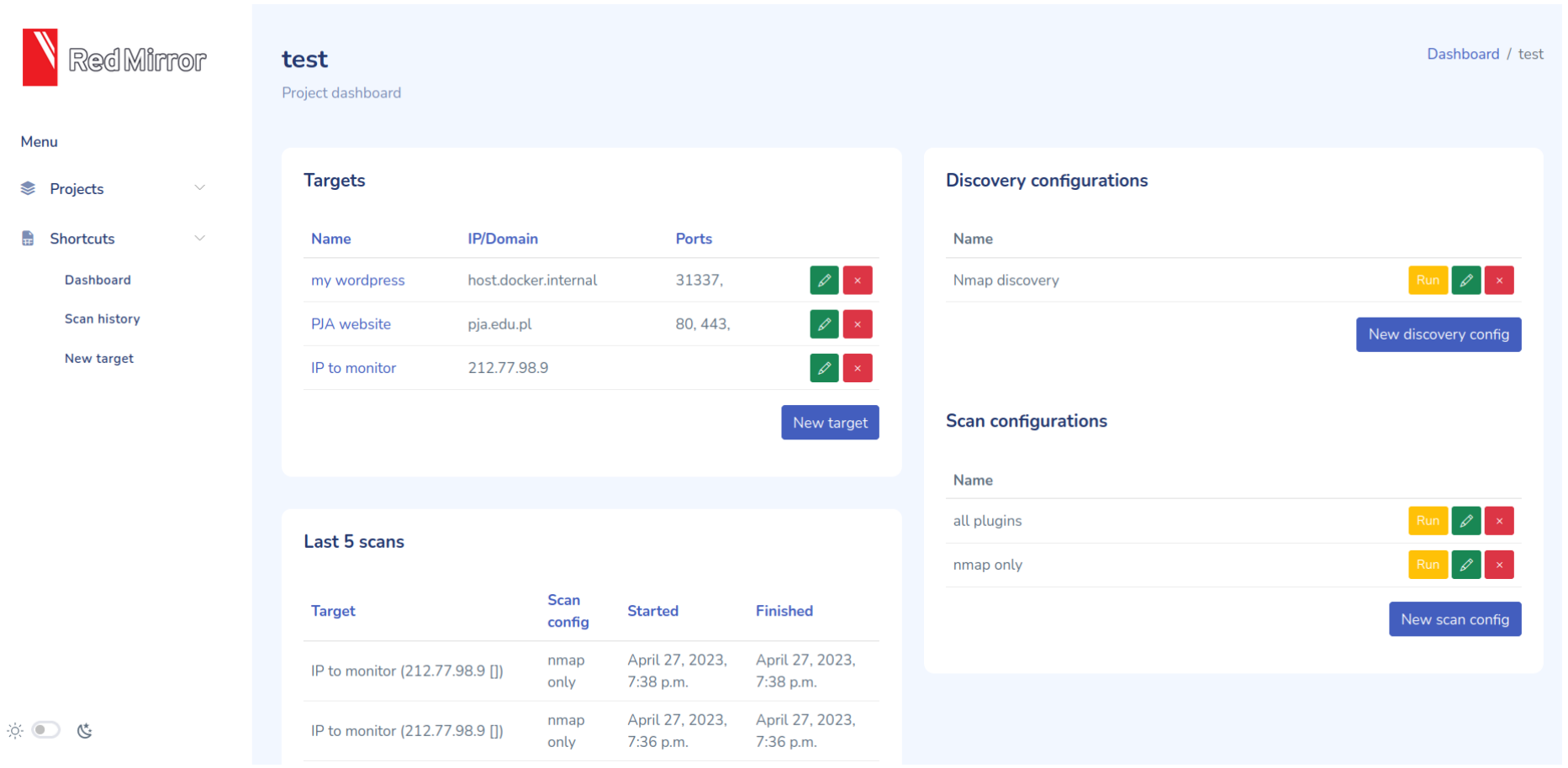
Rysunek 5.5 przedstawia ekran z listą podatności pojedynczego serwisu w Celu. Z tego miejsca można dokonać edycji ryzyka podatności (Rysunek 5.6) lub przejść do tekstowego wyniku skanu, który wykrył daną podatność.

Rysunek 5.7 pokazuje ekran historii skanów. Każda sekcja to oddzielna grupa skanów. Na tym ekranie można monitorować postęp skanowania oraz przejść do ich rezultatów.

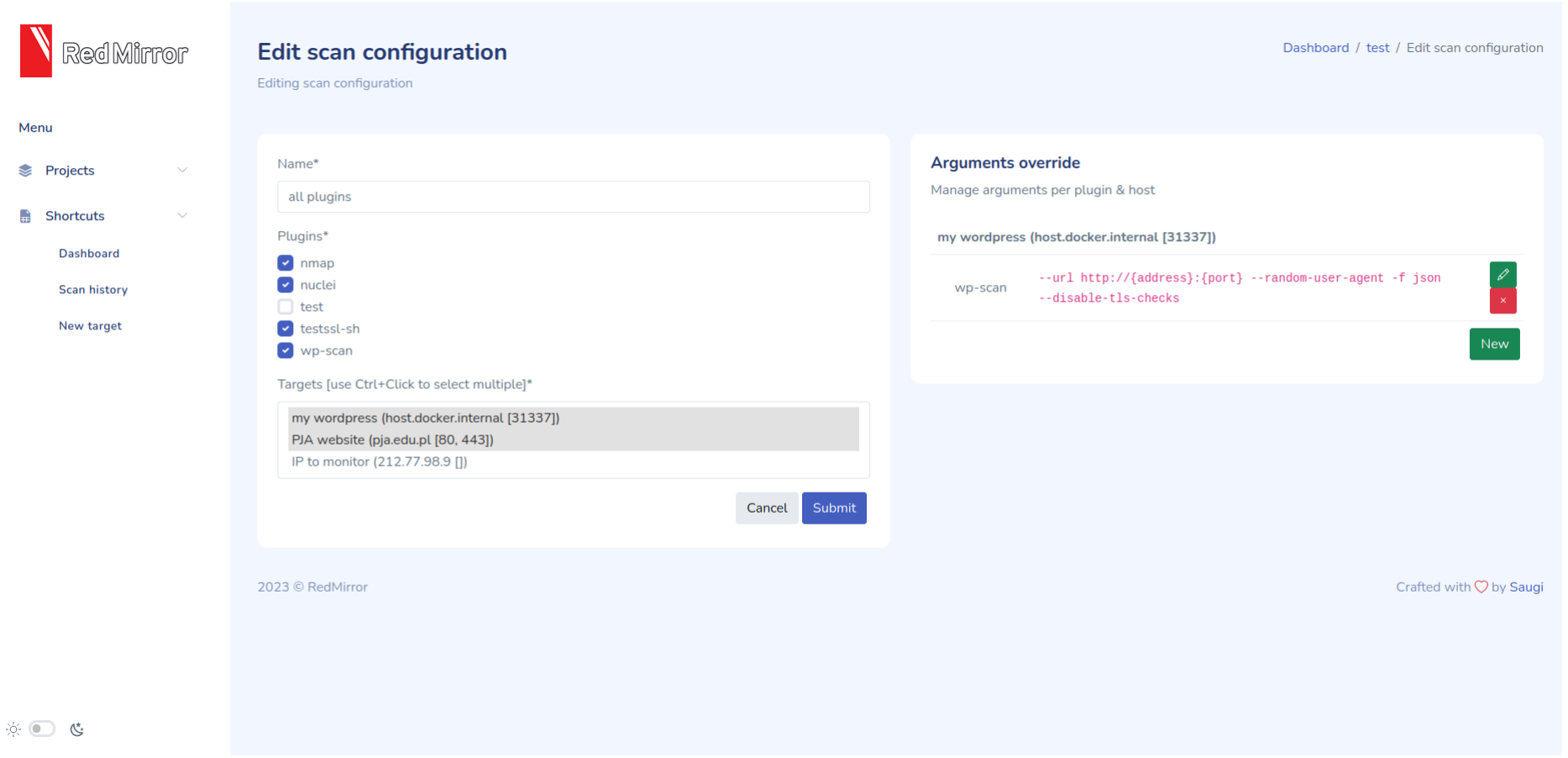
Rysunek 5.8 przedstawia ekran rezultatów grupy skanów (konfiguracji skanu uruchomionej na określonych *Celach*). Są na nim widoczne wtyczki użyte na każdym z Celów, które po rozwinięciu pokazują wynik działania narzędzia w formie tekstowej.



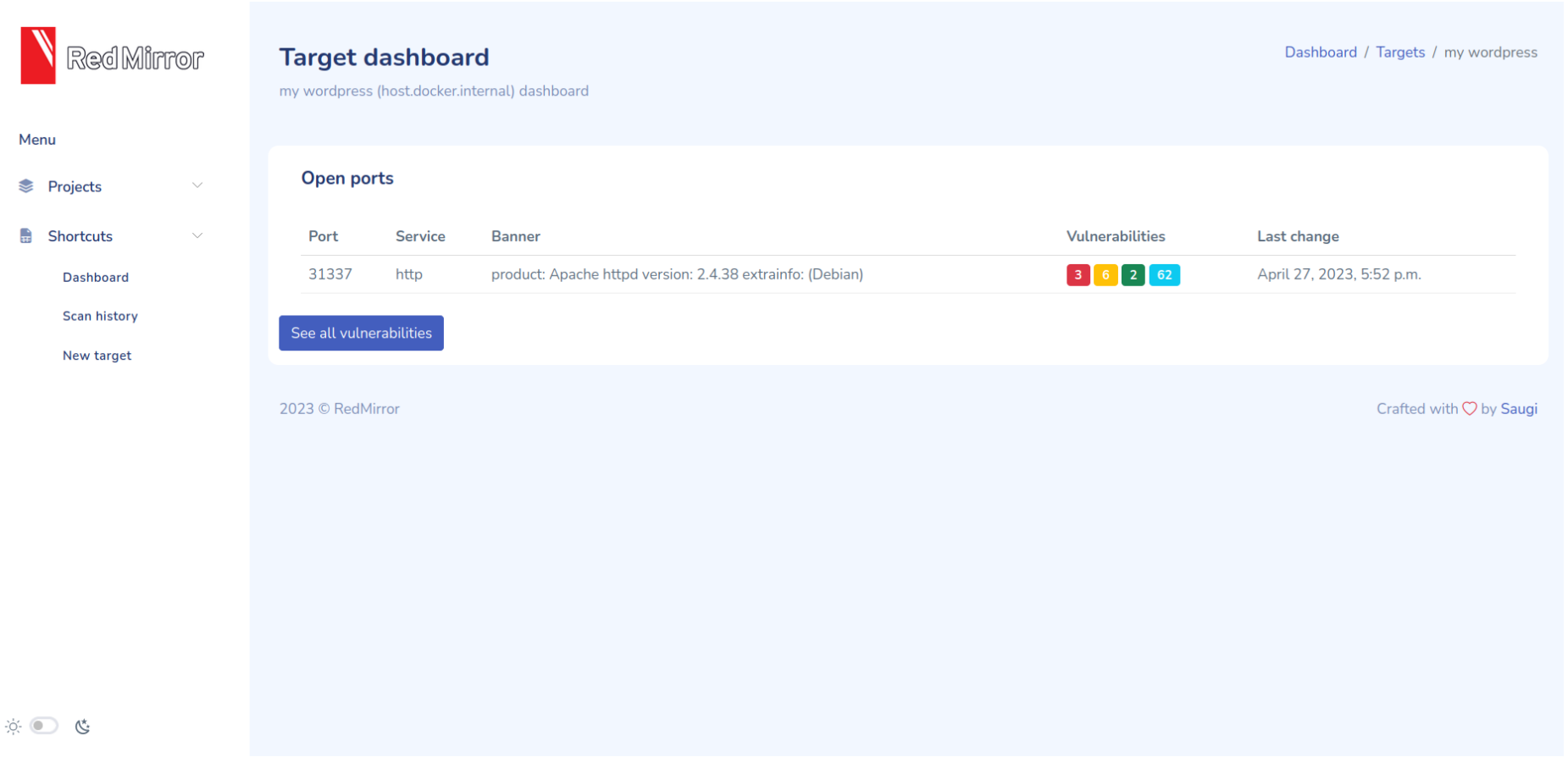
Rysunek 5.1: Projekty w RedMirror



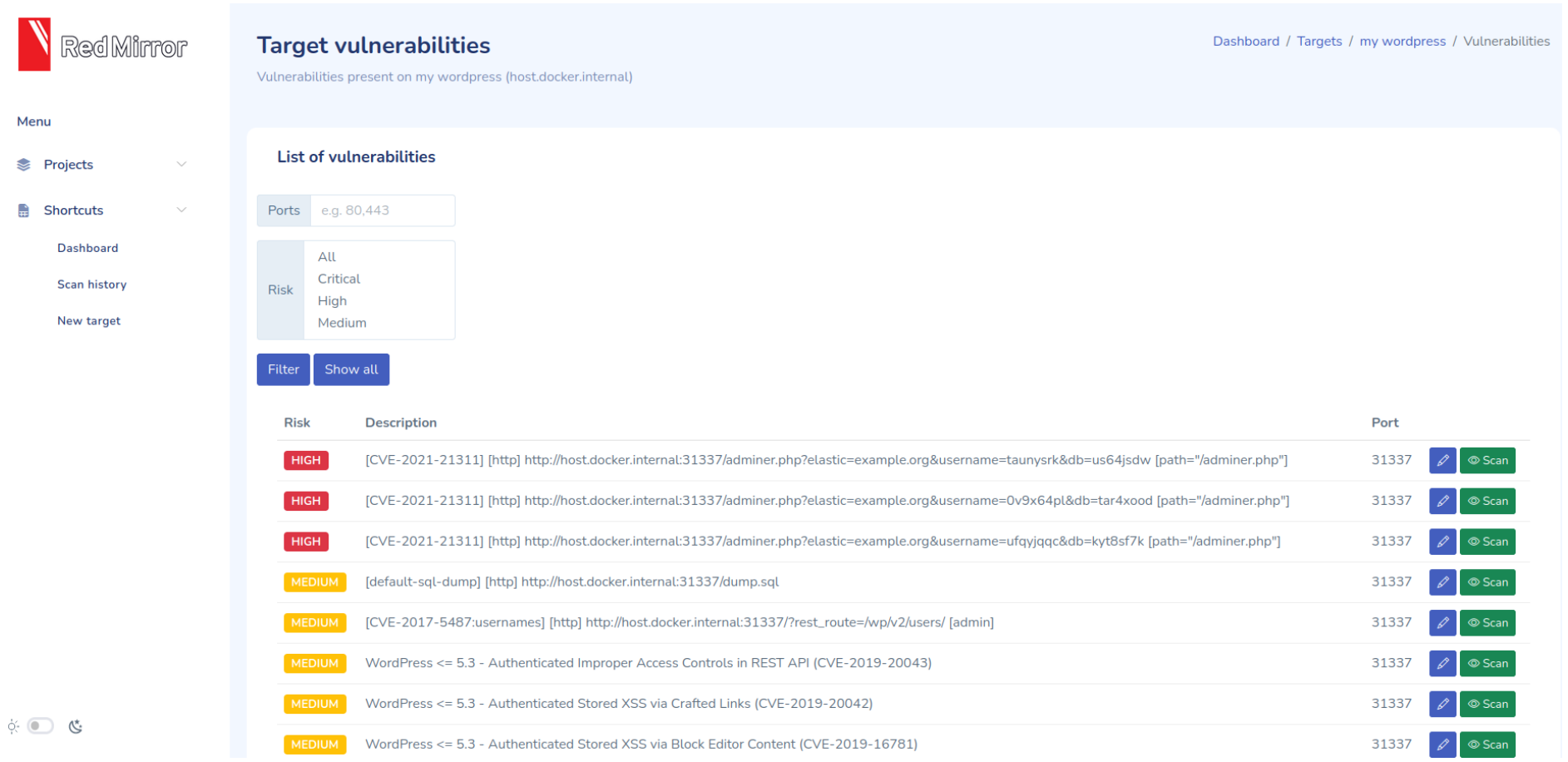
Rysunek 5.2: Dashboard projektu



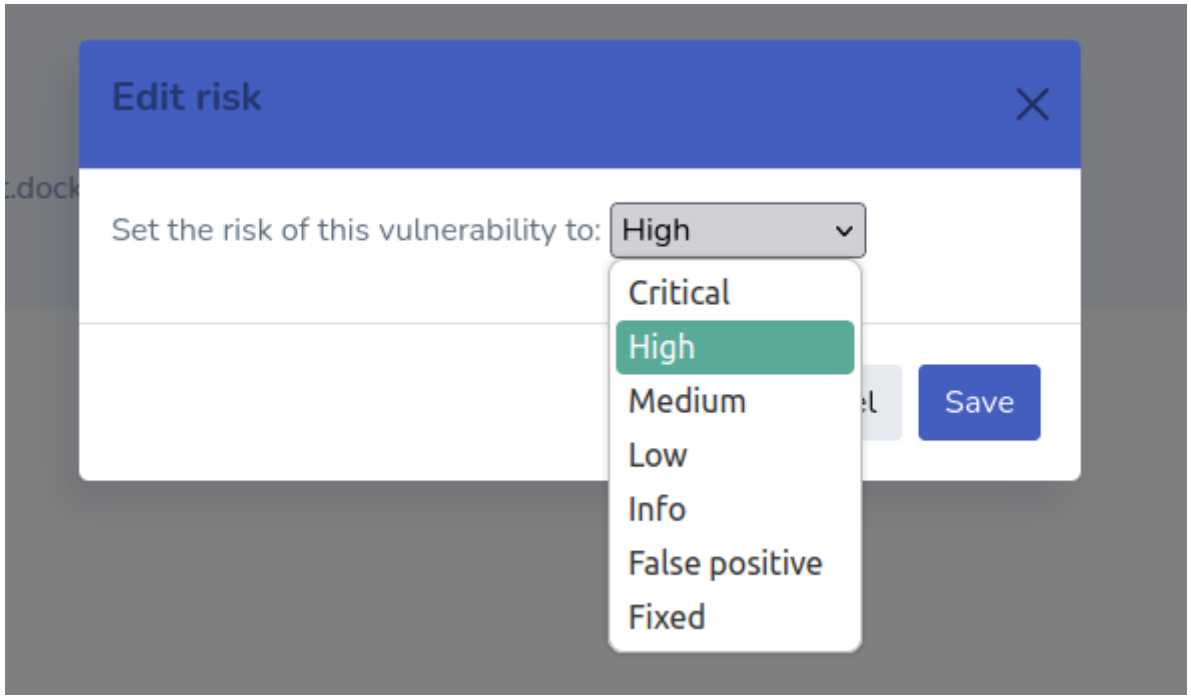
Rysunek 5.3: Edycja konfiguracji skanu



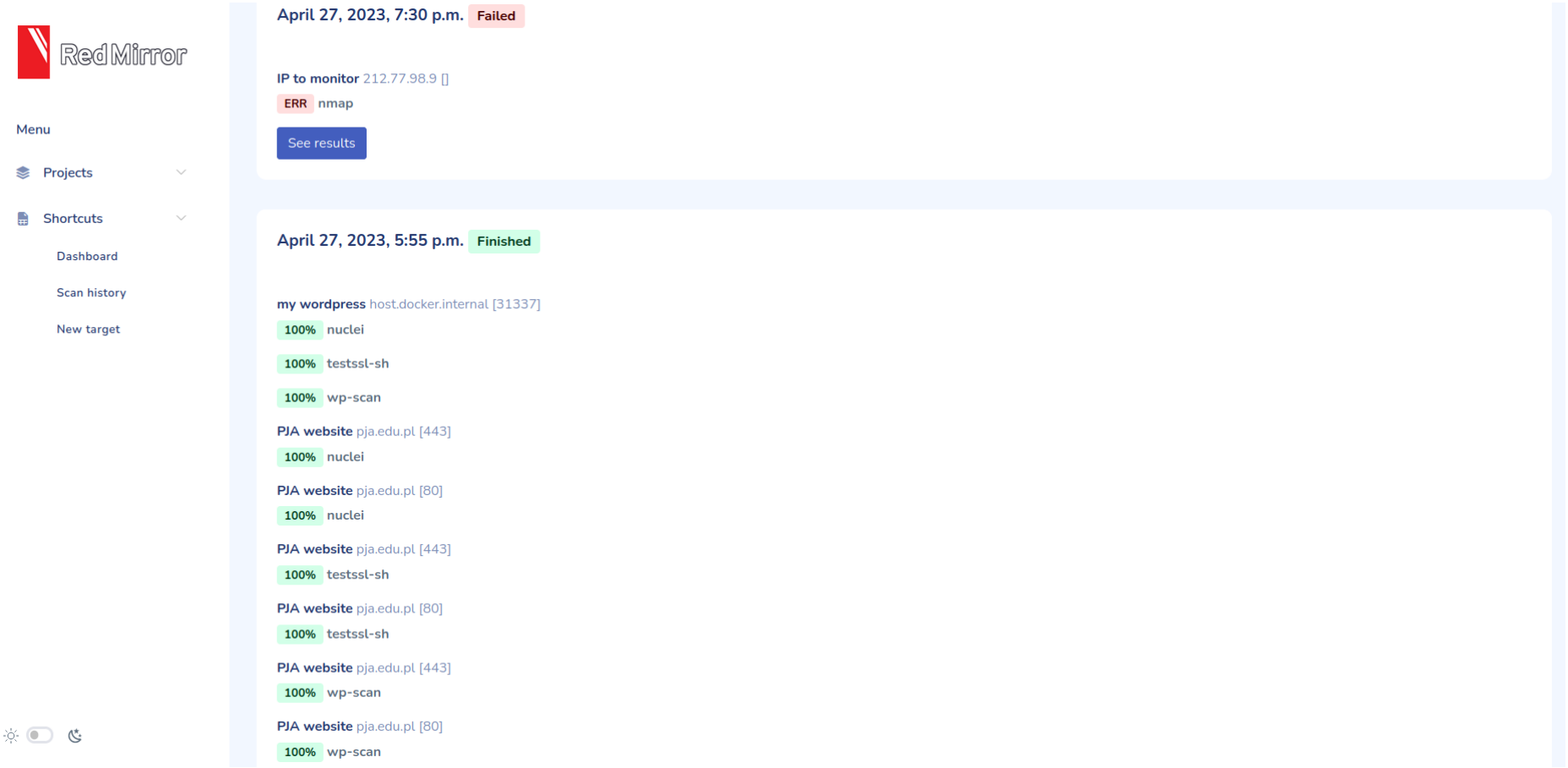
Rysunek 5.4: Dashboard Celu



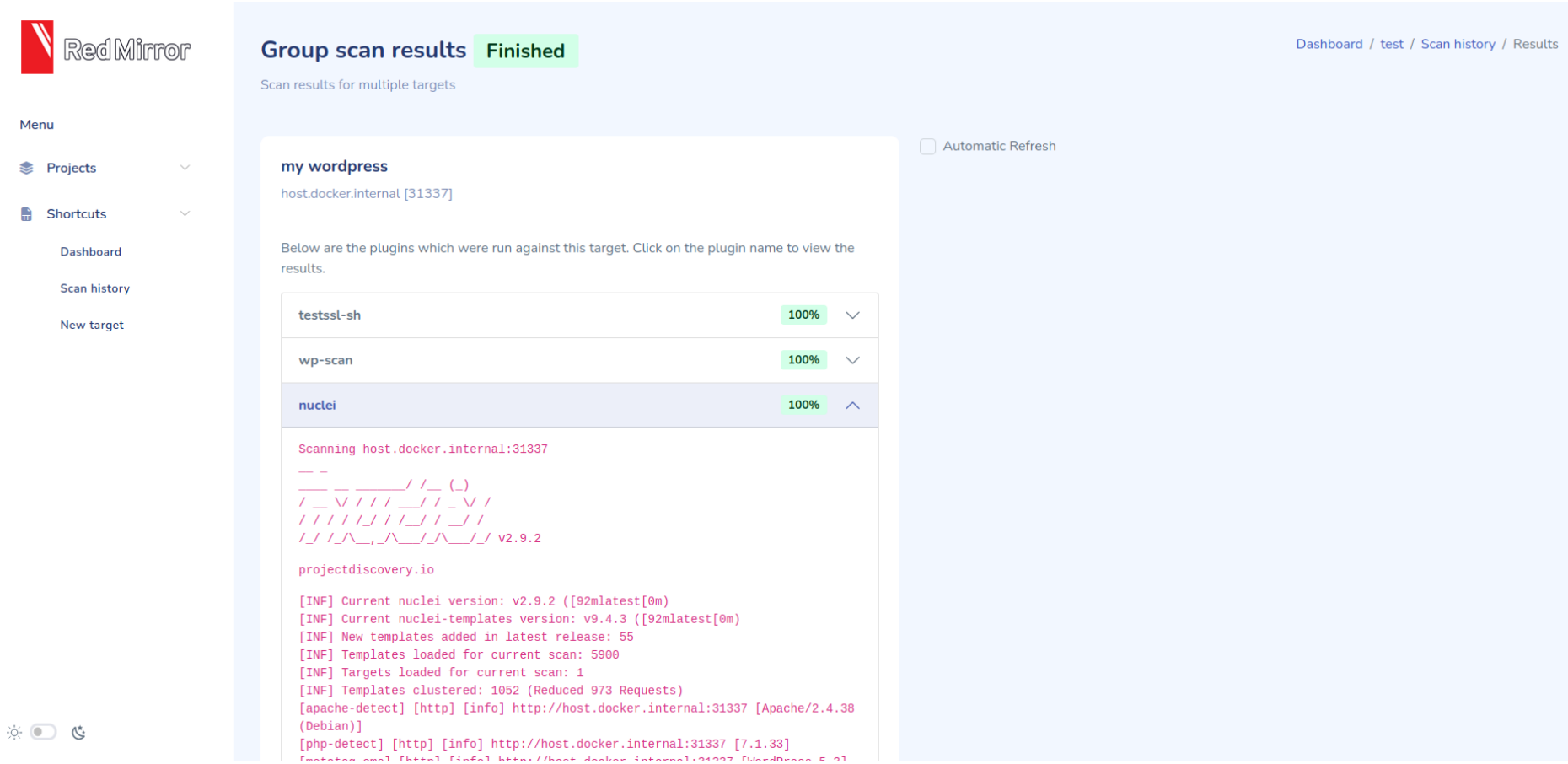
Rysunek 5.5: Serwisy i podatności Celu



Rysunek 5.6: Edycja ryzyka podatności



Rysunek 5.7: Historia skanów



Rysunek 5.8: Wyniki grupy skanów

5.2 Wybór technologii

Z uwagi na charakter narzędzia, priorytetem w wyborze technologii była jego popularność, a tym samym wsparcie społeczności. Gdyby narzędzie było zamknięte (bez publicznie dostępnego kodu źródłowego), technologia nie miałaby takiego znaczenia, ale w projekcie otwartym warto zmniejszać próg wejścia osobom, które mogą chcieć wprowadzić zmiany, poprawki czy ulepszenia. Wybór technologii powinien być także bardziej konserwatywny: nowe projekty mogą być mniej stabilne oraz znane przez użytkowników.

5.2.1 Język programowania

W przypadku aplikacji związanych z testami bezpieczeństwa, oczywistym wyborem języka programowania jest *Python*, ponieważ jest on szeroko używany w środowisku hakerów [53]. Wiele nowych narzędzi i exploitów jest pisana właśnie w *Pythonie*, co może ułatwić ich integrację z rdzeniem narzędzia. Python jest także językiem przystępnym, co dodatkowo ułatwia pracę dla potencjalnych użytkowników - testerzy bezpieczeństwa mogą nie być programistami, zatem ich umiejętności nie pozwolą na użycie bardziej skomplikowanych języków programowania lub to skutecznie utrudnią. Dodatkowo, Python wymaga mało pracy jeśli chodzi o jego instalację na systemach operacyjnych, a w wielu dystrybucjach Linuxa jest także wbudowany [74].

5.2.2 Framework do aplikacji webowej

Sterowanie narzędziem odbywa się poprzez aplikację webową. Do jej stworzenia - kierując się zasadą, że do oprogramowania *open-source* warto wybrać najprostsze rozwiązanie - użyto frameworka *Django*. Wbudowane mechanizmy w *Django* oraz mnogość ogólnodostępnych modułów pozwala na stworzenie prostej, zrozumiałej dla mniej doświadczonych developerów aplikacji webowej. Koncepcja z założeń ma służyć jednemu lub kilku użytkownikom, zatem perspektywa wydajności czy oszczędności zasobów nie jest istotna. Domyślnie *Django* działa w podejściu *Model-View-Template (MVT)*, co ułatwia tworzenie aplikacji dla osób nieznających języka *JavaScript*. Zamiast tworzyć oddzielne aplikacje (frontendową i backendową), w *Django* można stworzyć jedną aplikację, która zawiera komponenty związane z systemem, oraz szablony, do których przesyłane są dane z tych pierwszych. Same szablony pisano z użyciem *Jinja2* - również bardzo popularnego języka do tworzenia szablonów HTML z elementami dynamicznymi integrującymi się z *Django*. Jest to alternatywa dla szablonów wbudowanych w *Django*. Nie było jednak konkretnego powodu dlaczego szablony były pisane z użyciem tego silnika.

5.2.3 Frontend

Do strony frontendowej, czyli GUI aplikacji, wykorzystano *Mazer*, który jest gotowym panelem administracyjnym bazującym na *Bootstrap 5*[75]. Wybrano to konkretne rozwiązanie ze względu na możliwość szybkiego stworzenia warstwy wizualnej dla użytkownika. *Mazer* oraz *Bootstrap 5* zawierają bardzo dużo gotowych komponentów, dzięki czemu oszczędza się czas na pisaniu stylów, a całe strony można bazować na przygotowanych przez twórców przykładach.

Istotną częścią pracy z motywem było podzielenie go na części powtarzalne (np. nagłówki, stopka i menu) oraz unikalne (treść strony). Miało to na celu zlikwidowanie potrzeby umieszczania niemal tych samych elementów w każdym pliku szablonu. Subtelne, dynamiczne zmiany takie jak np. zmiana tytułu czy też podświetlenie elementu menu bazując

na aktualnie otwartej stronie, osiągnięto wykorzystując dostępne i samodzielnie zdefiniowane tagi w szablonach *Jinja2*.

5.2.4 Konteneryzacja

Jak wspomniano w punkcie Aktualizacje narzędzi, kontenery są przydatnym narzędziem do rozwiązania problemów zapewnienia zgodności wersji narzędzi oraz utrzymania stabilności rozwiązania. Jako metodę konteneryzacji wybrano *Docker*, który jest najpopularniejszym rozwiązaniem na świecie [76] i posiada duże wsparcie społeczności. W założeniach projektu nie stwierdzono szczególnych potrzeb, zatem wybór technologii nie musiał się opierać o analizę dostępnych w nich funkcji.

5.3 Architektura

Architektura rozwiązania była najbardziej istotną kwestią w tym projekcie. Wielokrotnie założenia były redefiniowane ze względu na pojawiające się problemy lub oznaki, że mogą one pojawić się w przyszłości. Wprowadzanie zmian architektonicznych w aplikacji w czasie jej używania byłoby bardzo uciążliwe, ponieważ wymagałoby obsłużenia wykorzystywanych już danych (np. poprzez skrypt przekształcający dane ze starej wersji do nowej) lub podjęcia decyzji o konieczności wprowadzania konfiguracji na nowo. Dodatkowo, chcąc zachować generyczność aplikacji, rozwiązanie powinno być możliwie elastyczne, aby obsłużyć sytuację każdego lub przynajmniej większości użytkowników. Wiele z teoretycznych problemów zostało już opisanych i rozwiązanych w sekcji Problemy i rozwiązania w rozdziale Generyczny skaner bezpieczeństwa. Ten rozdział skupi się zatem na problematyce samej implementacji koncepcji.

5.3.1 Diagram klas

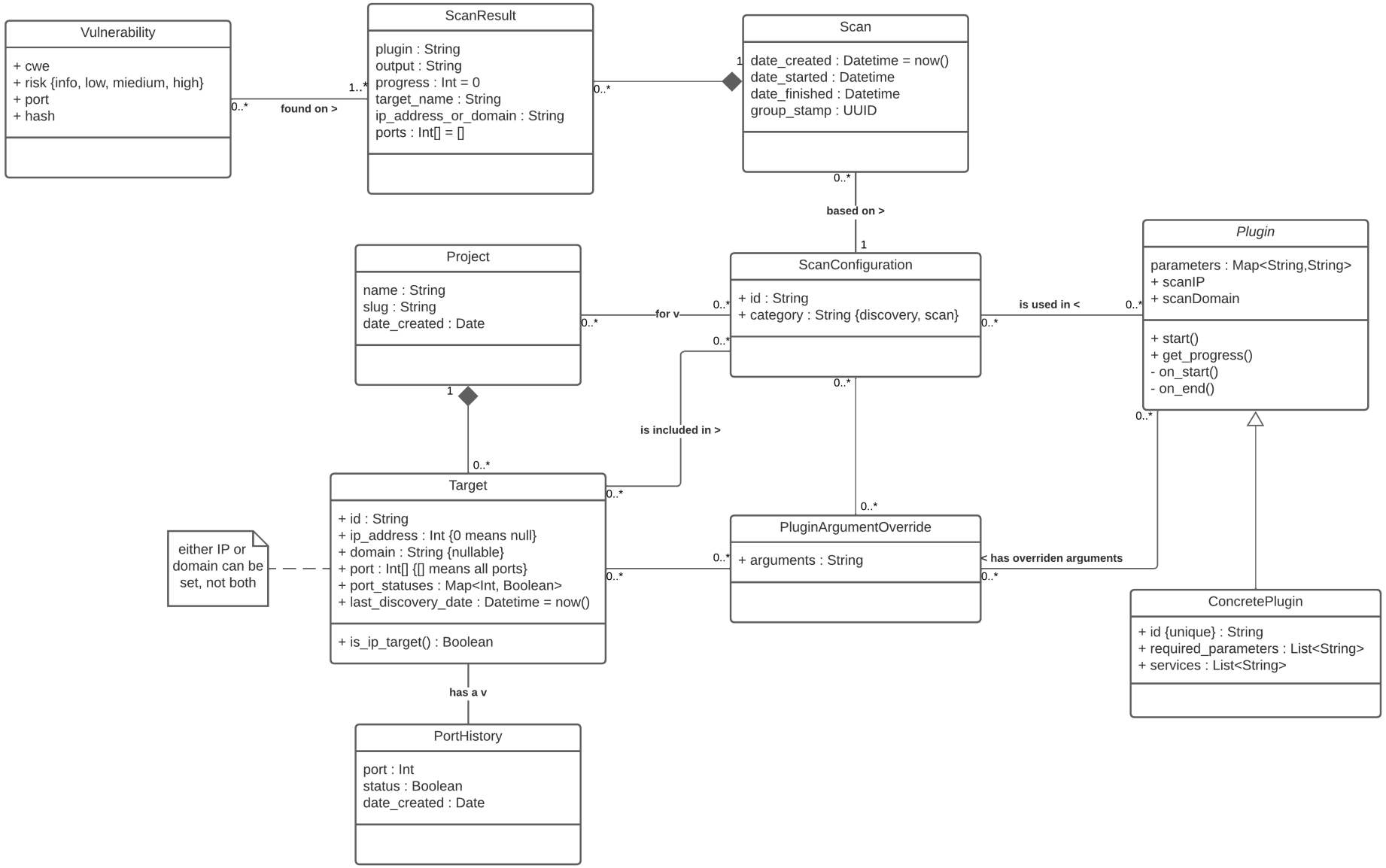
Diagram klas to rodzaj diagramu strukturalnego używanego w projektowaniu i opisie systemów informatycznych. Wykorzystywany jest przede wszystkim w kontekście języka modelowania UML (*Unified Modeling Language*), ale jego zastosowanie jest szersze i może być on używany w innych kontekstach.

Diagram klas opisuje strukturę systemu poprzez pokazanie systemu klas, ich atrybutów, metod (operacji), a także relacji między nimi. Klasa to podstawowy koncept w obiektowo zorientowanych językach programowania, a diagram klas to narzędzie umożliwiające przedstawienie, jak te klasy są powiązane i jak mogą współpracować[77]. Podstawowe elementy diagramu klas to:

- **Klasy** - przedstawione jako prostokąty podzielone na trzy części: nazwa klasy, atrybuty i metody.
- **Relacje** - przedstawione jako linie łączące klasy. Istnieje kilka rodzajów relacji, w tym asocjacje, dziedziczenie, realizacje interfejsu, zależności i agregacje.

Diagram klas pomaga zrozumieć strukturę kodu źródłowego, pokazuje, jak różne części systemu są ze sobą powiązane oraz pomaga identyfikować potencjalne problemy w projektowaniu systemu [77].

Rysunek 5.10 prezentuje diagram klas rozwiązania *RedMirror*, na podstawie którego powstała pełna implementacja projektu. Implementacja w Pythonie nieznacznie różniła się detalami takimi jak typy danych, ale ogół rozwiązania nie odbiega od tego, co zostało zaprojektowane.



Rysunek 5.9: Diagram klas RedMirror

5.3.2 Diagram sekwencji

Rysunek 5.10 prezentuje diagram sekwencyjny procedury wykonania *Konfiguracji skanu*.

5.3.3 Ogólna budowa aplikacji

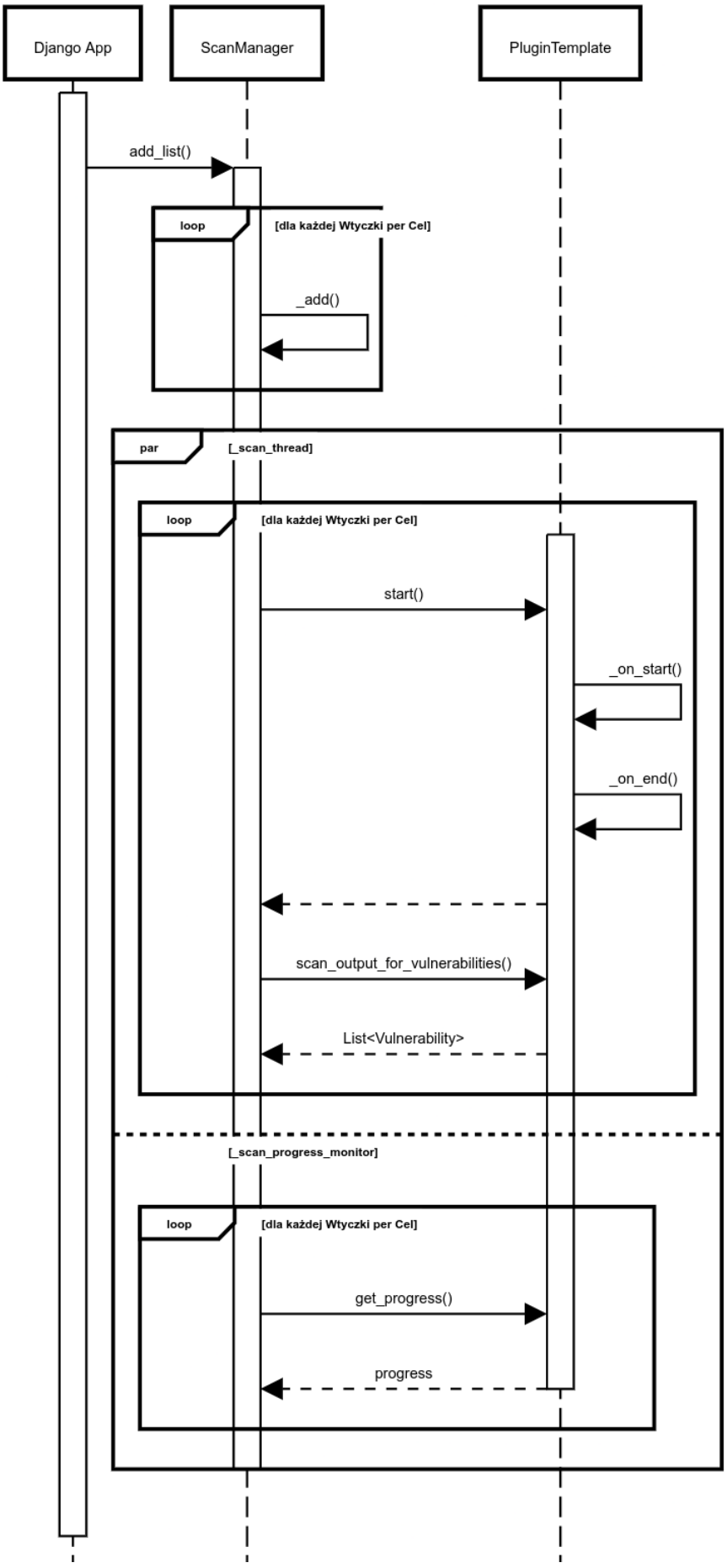
Szkielet aplikacji został stworzony na bazie szkieletu do aplikacji webowych tworzonych w *Django*. Korzystano także z rekomendacji opisanych w repozytorium *Django-Styleguide* dostępnych na pod adresem [78].

Struktura aplikacji

- RedMirror - konfiguracja *Django*, przekierowanie do głównego folderu aplikacji
- scanner - główny folder aplikacji
  - dao - obsługa odczytu/zapisu bazy danych
  - logic - logika aplikacji
  - migrations - migracje bazy danych
  - models - modele; w głównym katalogu znajdują się modele z bazy danych
    - logic - modele używane tylko przez logikę
    - view - modele używane przez widoki
  - plugins - wtyczki
    - discovery - wtyczki typu *discovery*
    - repo - wtyczki typu *scan*
    - utilities - funkcjonalności do ułatwiania zadań wtyczkom
  - static - pliki części frontendowej aplikacji, szablon Mazer
    - scanner
      - assets
      - css
      - js
      - mazer
  - templates - szablony Jinja2 wykorzystywane w aplikacji
    - scanner
      - base
      - projects
  - templatetags - tag do oznaczania aktywnego elementu w menu
  - views - funkcje aplikacji webowej (główna część aplikacji)

5.3.4 Modele w projekcie

Modelami są klasy, których główną funkcją jest przechowywanie danych. Dane z modeli są zapisywane w bazie danych lub na podstawie danych z bazy tworzone są modelowe obiekty. Ze względu na użycie ORM (*Object-Relational Mapping*), klasy są tożsame z encjami w bazie



Rysunek 5.10: Diagram sekwencji wykonania *Konfiguracji skanu*

danych. Z poziomu *Pythona* w modelach ustala się także relacje pomiędzy encjami (klucze obce), typy pól, niektóre reguły walidacyjne oraz sposób zarządzania rekordami w przypadku usunięcia rekordów połączonych kluczem obcym.

Aby zapewnić zgodność treści z kodem, będą używane nazwy angielskie - takie, jakie zostały użyte w kodzie. W podsekcjach podane są także definicje klas z zawartymi polami. Gwiazdką oznaczone są referencje do oddzielnych klas (klucze obce patrząc od strony bazy danych).

`Project(name, slug, date_created)`

Jest to klasa służąca jako miejsce do przypisania różnych celów. W realnym scenariuszu *Projektem* może być np. *SZKOŁA*, w której *Celami* będą komputery w niej dzialające.

`Target(name, ip_address_or_domain, ports[], project*, last_discovery_date, open_ports[])`

Target po zdefiniowaniu zawiera swoją nazwę oraz adres IP albo domenę. W implementacji domena/IP jest przechowywana w tej samej zmiennej. Dopiero później moduły będą ustalać, czy korzystają z domeny czy z adresu IP. Lepiej jest jednak podać domenę, ponieważ kilka domen (np. z różnymi stronami internetowymi) może być przypisane do jednego adresu IP, co jednocześnie powiększy powierzchnię skanowania. Z adresu domenowego można w prosty sposób uzyskać adres IP, ale w drugą stronę może to stanowić problem. Do zweryfikowania, czy podany *string* jest adresem IP czy domeną, służy funkcja `ip_address_or_domain`.

`ScanConfig(name, category, plugins[], *project, *targets)`

Do przeprowadzenia skanu tworzona jest konfiguracja. W konfiguracji ustala się, jakie wtyczki mają zostać uruchomione na jakim hoście. Dodatkowo ustala się typ konfiguracji - *discovery* lub *scan*. *ScanConfig* może (najprawdopodobniej chwilowo) nie posiadać żadnego przypisanego *Celu*, dlatego musi posiadać informację o tym, do jakiego projektu odnosi się dana konfiguracja. Nie może być ona użyta poza projektem.

`PluginArgumentOverride(arguments, plugin, target*, scan_config)`

Ta klasa służy do zastąpienia domyślnych argumentów przekazywanych do uruchomienia modułu w określonej konfiguracji skanu. W celu zapewnienia maksymalnej generyczności, nowe argumenty są ustalane dla pojedynczej wtyczki i dla pojedynczego celu. Nie można nadpisać argumentów ponownie dla tej samej wtyczki i *Celu* - należy zmodyfikować lub usunąć poprzedni obiekt.

`Scan(date_created, date_started, date_finished, *target, *scan_config, group_stamp)`

Scan to uruchomienie jednej *Konfiguracji skanu* na jednym *Celu*. W przypadku, kiedy *Konfiguracja skanu* zawiera więcej *Celów*, utworzone zostanie tyle obiektów *Scan*, ile jest *Celów*. Takie podejście ma na celu ułatwienie monitorowania etapów skanowania poprzez pola *date_created*, *date_started* i *date_finished*. Pole *group_stamp* służy do znakowania skanów, które były uruchomione *jednocześnie* poprzez uruchomienie konkretnej *Konfiguracji skanu*. Bazując na *Konfiguracji skanu*, nie można jednak grupować elementów przeglądając historię, ponieważ *Konfiguracja skanu* może się w międzyczasie zmienić. Dlatego przy uruchomieniu *Konfiguracji skanu* wszystkie utworzone w tej *sesji* obiekty *Scan* będą miały ten sam *group_stamp*.

ScanResult(*scan, plugin, output, progress, target_name, ip_address_or_domain, ports[])

To klasa, której obiekty przechowują informacje o uruchomieniu pojedynczej wtyczki na pojedynczym *Celu*. Tutaj pojawia się postęp w działaniu oraz rezultaty (*output*) skanowania. Z uwagi na to, że obiekty klasy *Target* są edytowalne, informacje o skanowanych obiektach są także zapisywane w tej klasie jako oddzielne pola. W pierwotnych założeniach *ScanResult* miał być pojedynczym uruchomieniem wtyczki na pojedynczym *Celu*, ale w obecnej wersji jest to rozdrobnione jeszcze bardziej: skan odbywa się na pojedynczym porcie. Nie dotyczy to jednak skanów z wykorzystaniem wtyczek *discovery*, które wykonywane są na wielu portach na raz. Należy zaznaczyć, że *ScanResult* to, w prostych słowach, najmniejszy obiekt reprezentujący skanowanie w aplikacji. Nazwa także może być myląca, ponieważ *ScanResult* nie zawsze jest rezultatem skanowania - *ScanResult* w większości przypadków najpierw jest najmniejszą częścią zlecenia skanowania i dopiero później rezultat jest w nim zapisywany. Teoretycznie, zlecenie z niepełnymi rezultatami mogłoby być oddzielnym obiektem niż obiekt sugerujący, że jest rezultatem, natomiast nie było potrzeby, aby ten pierwszy istniał po zakończeniu skanowania - wszystkie rezultaty i tak będą zapisane w bazie danych.

PortHistory(date_created, *target, port, status, service, banner)

PortHistory to *kontener* do przechowywania informacji o statusie portu. Obiekty tej klasy można traktować podobnie jak logi, ponieważ są niezmienne. Ma to na celu przetrzymywanie historii zmian statusów i działających serwisów na danym porcie. Z kolei, aby otrzymać możliwie najnowszą informację o statusie portu lub działającym na nim serwisie, należy sugerować się najwyższą datą (*date_created*).

Vulnerability(RISK_LEVELS, hash, cwe, additional_description, risk, port)

Vulnerability to klasa, której obiekt opisuje unikalną podatność, która wystąpiła w serwisie. *RISK_LEVELS* to stała tablica, która zawiera możliwe ustawienia ryzyka podatności. Podatność musi zawierać port, na którym została znaleziona, oraz jej ryzyko. Z opcjonalnych elementów podatność może zawierać opis (*additional_description*) i numer CWE. Pole *hash* należy ustawiać funkcją *generate_hash* dostępną w klasie. Liczy ona sumę kontrolną podatności na podstawie id *Celu*, numeru CWE, opisu oraz portu. Taki hash jest używany w celu ułatwienia porównywania ze sobą podatności (ważne jest tylko to, czy podatność jest taka sama czy nie) oraz jako jej unikalny identyfikator.

VulnerabilityOccurence(*vulnerability, *scan_result)

To klasa, której obiekty mapują unikalną podatność z rezultatem skanu. Dzięki tej klasie podatności po każdym skanowaniu nie są zduplikowane. Przypisuje ona także podatności do konkretnego *ScanResult* (uruchomieniu pojedynczej wtyczki na pojedynczym *Celu*), dzięki czemu po kluczach obcych można znajdować powiązane podatności ze skanami.

5.3.5 Wtyczki

Wtyczki umieszczone są w projekcie w katalogu `scanner/plugins`. W tym katalogu wtyczki typu *discovery* znajdują się w kolejnym katalogu `discovery`, zaś wtyczki typu *scan* w `repo`. Każda wtyczka dziedziczy po klasie abstrakcyjnej *PluginTemplate* (plik `plugin_template.py`) w katalogu `repo`, którą prezentuje Listing 5.1.

Listing 5.1: Kod klasy `PluginTemplate`

```
class PluginTemplate:
    name = ""
    # services should base on ↙
        ↳ https://github.com/nmap/nmap/blob/master/nmap-services
    services = []

    def start(self):
        print('Plugin start')
        self._on_start()
        self._on_end()

    @abstractmethod
    def get_progress(self):
        raise NotImplementedError

    @abstractmethod
    def get_output(self):
        raise NotImplementedError

    @abstractmethod
    def _on_start(self):
        raise NotImplementedError

    @abstractmethod
    def _on_end(self):
        raise NotImplementedError

    @abstractmethod
    def scan_output_for_vulnerabilities(self):
        raise NotImplementedError
```

Każda wtyczka dziedziczy po klasie *PluginTemplate* i musi dostosować się do już zadeklarowanych metod. Wtyczki nie mają swoich unikalnych nazw (nie licząc pola *name* - każda klasa nazywa się *Plugin*, co jest dozwolone w *Pythonie* i w tym przypadku nie rodzi problemów. Szczegóły dotyczące importu są opisane w sekcji Import wtyczek.

5.3.5.1 Opis metod

Metoda `start` jest wywoływana przez wątek stworzony przez *ScanManager* i uruchamia działanie wtyczki. Ta metoda zawiera prosty cykl życia, czyli wywołanie na sobie metod *_on_start* oraz `_on_end`. Te metody są implementowane przez twórcę wtyczki. O ile metoda `_on_start` jest kluczowa, ponieważ tam będzie się uruchamiała wtyczka, to metoda `_on_end` jest jedynie dodatkiem i lekkim udogodnieniem w celu zapewnienia, że instrukcje końcowe się wykonają nawet w przypadku błędu w metodzie `_on_start`.

W trakcie działania wtyczki *Rdzeń* aplikacji odpytuje o obecny postęp *Skanu*. Używa on na obiekcie wtyczki metody `get_progress`, która w idealnym scenariuszu zwraca aktualny stan działania wykorzystywanego narzędzia. Nie zawsze możliwe jest podanie konkretnego wyniku, ale należy przynajmniej stosować wartości 0 (jeszcze nieuruchomiony), dowolną wartość w zakresie 1-99 (w trakcie) i 100 (gdy narzędzie zakończy działanie). W przypadku wystąpienia błędu należy ustawić wartość -1.

Metoda `get_output` służy do zwrócenia aktualnego wyniku działania narzędzia. W typowym scenariuszu *output* narzędzia jest na bieżąco dopisywany do zmiennej we wtyczce. *ScanManager* co jakiś czas odpytuje wtyczkę tą metodą, aby aktualny wynik był zapisywany w bazie danych. Jest to jednak jedynie wynik tekstowy, czyli to, co zwraca narzędzie (zakładając, że jest to narzędzie konsolowe i zwraca rezultaty w formie tekstu).

Po zakończeniu działania narzędzia, *ScanManager* wywoła na obiekcie wtyczki metodę `scan_output_for_vulnerabilities`. Ta metoda powinna być zaimplementowana przez twórcę wtyczki, aby znalezione podatności były zapisane w bazie danych i przypisane do konkretnych serwisów skanowanego hosta. Metoda najczęściej będzie musiała sparsować zapisany w zmiennej *output* narzędzia i przekształcić go na listę podatności - dokładniej obiekty klasy *VulnerabilityFromPlugin*, który prezentuje Listing 5.2.

Listing 5.2: Kod klasy `VulnerabilityFromPlugin`

```
class VulnerabilityFromPlugin:
    def __init__(self, additional_description: str, risk: str, port: int, ↙
        ↳ cwe: str = None, scan_result: ScanResult = None):
        if cwe is None:
            cwe = ''
        if additional_description is None:
            additional_description = ''

        self.additional_description = additional_description
        self.risk = risk
        self.port = port
        self.cwe = cwe
        self.scan_result = scan_result
```

5.3.5.2 Import wtyczek

Wtyczki w aplikacji są ładowane w trakcie jej działania, ale tylko raz. Wtyczki są importowane ze wspomnianych wcześniej katalogów. Do listowania obecnych w projekcie wtyczek wykorzystana jest klasa *PluginHelper*, która jest *singletonem* stworzonym do zarządzania wtyczkami. Przy tworzeniu instancji klasy zbiera ona istniejące wtyczki (metoda *_walk_package*), które dziedziczą po klasie *PluginTemplate*. Każda z wtyczek jest importowana poprzez bibliotekę *importlib* i zapisywana w liście *_plugin_modules*. Każda klasa wtyczki ma nazwę *Plugin* i nie tworzy to żadnych konfliktów dla aplikacji. Klasy są tworzone tylko w celu bycia zaimportowanym przez *PluginHelper*, który trzyma referencję do klasy, której obiekt można utworzyć później.

Klasa zawiera także inne metody, które mają na celu zwracanie informacji o zaimportowanych wtyczkach (*list_plugin_names*, *list_plugin_names_with_default_arguments*) lub o serwisach obsługiwanych przez wtyczki (*get_services_by_plugin_name*, *get_services_by_plugin_names*). *PluginHelper* działa także jako fabryka tworząca instancje wtyczki poprzez metodę *get_plugin_instance_by_name*.

5.3.6 Zarządzanie skanami

Zlecenie skanu odbywa się z poziomu GUI, czyli w przypadku tej implementacji, z aplikacji webowej. Zlecenie uruchomienia skanowania (kliknięcie w GUI) wysyłane jest do punktu końcowego aplikacji webowej `scan_config_run`, który jako argument przyjmuje jedynie ID *Konfiguracji skanu*. W tej funkcji kontrolera tworzona jest lista *Celów* do skanowania, a następnie przesyłana do obiektu *ScanManager*.

ScanManager jest obiektem, który zajmuje się zarządzaniem trwającymi i zleconymi skanami. Może istnieć tylko jedna instancja klasy *ScanManager*. Aby to zapewnić, został użyty wzorzec projektowy *Singleton*, korzystając z mechanizmu metaklas w Pythonie [79]. *ScanManager* ma 4 metody:

- **`_add`**

  : Przyjmuje pojedyncze zlecenia skanowania - argumentem jest obiekt typu *ScanResult*.

- **`add_list`**

  : Przyjmuje listę *Skanów* wraz z ich kategorią (*discovery* lub *scan*). Zarządza tworzeniem grupy skanów (ustalenie *group_stamp*, przypisanie odpowiednich wtyczek do serwisów, utworzenie obiektów *ScanResult* oraz przesłanie każdego z nich do metody `_add`.

- **`_scan_thread`**

  : Pierwszy wątek uruchamiany przez ScanManagera w celu zarządzania działającymi skanami. Obserwuje kolejkę skanów i w momencie pojawienia się w niej nowego obiektu *ScanResult*, uruchamia odpowiednie wtyczki. Działający skan jest przechowywany w obiekcie *RunningScanResult*, a wszystkie takie skany w oddzielnej liście. Jeśli konfiguracja skanu korzysta z nadpisywania argumentów, to zostaną one także przesłane do wtyczek. Po zakończeniu działania narzędzia wtyczki, opisany wątek prosi wtyczkę o przekazanie listy podatności, którą następnie wątek przesyła do *VulnerabilityDao* w celu jej zapisania.

- **`_scan_progress_monitor`**

  : Drugi wątek uruchamiany przez *ScanManager* w celu monitorowania postępu skanów. Odpytuje on kolejno działające skany (obiekty *RunningScanResult* przechowywane w liście) i aktualizuje rezultat ich prac w bazie danych. Dzięki temu można obserwować wynik pracy wtyczek jeszcze w czasie ich pracy. Ten wątek *finalizuje* także proces skanowania poprzez działania wykonywane w momencie, gdy narzędzie we wtyczce skończy pracę - poprawnie (gdy *progress* jest równy 100) lub poprzez błąd (gdy *progress* jest równy -1). Zapisuje on finalne dane do bazy danych wraz z datą zakończenia skanu.

5.3.7 Wykorzystanie bazy danych

Aplikacja korzysta z bazy danych do przechowywania kluczowych informacji niezbędnych do prawidłowego jej działania. Informacje te obejmują między innymi dane o Celach, konfiguracji oraz historii skanów. Wymóg zarządzania takimi danymi jednoznacznie wskazywał na konieczność zastosowania bazy danych jako miejsca do przechowywania danych. Zdecydowano się na wykorzystanie podejścia *Code-first*, korzystając z wbudowanych mechanizmów dostarczanych przez *framework Django*, który wykorzystuje podejście ORM (*Object-Relational Mapping*).

Metoda *Code-first* to strategia, w której struktura bazy danych jest generowana na podstawie kodu źródłowego aplikacji. W praktyce, zamiast tworzenia bazy danych za pomocą czystego SQL lub jakiegokolwiek języka zapytań, tworzy się klasy w Pythonie (modele), które reprezentują strukturę danych. *Django*, przekształca te definicje klas w struktury bazy danych, tworząc odpowiednie tabele i pola. ORM to technika, która pozwala na interakcję z danymi w bazie danych za pomocą obiektów i klas w języku programowania, zamiast pisania bezpośrednich zapytań SQL.

Kluczową częścią podejścia *Code-first* jest system migracji *Django*. Migracje są jak zestawy instrukcji dla bazy danych, które mówią jak stworzyć lub zmodyfikować schemat bazy danych. *Django* jest w stanie automatycznie generować migracje na podstawie różnic między obecnym stanem bazy danych a stanem zdefiniowanym przez modele w aplikacji.

Dzięki wykorzystaniu *Code-first* można efektywnie zarządzać schematem bazy danych bezpośrednio z poziomu kodu źródłowego aplikacji. Ułatwia to procesy takie jak rozwijanie aplikacji, utrzymanie jej i kontrolę wersji schematu bazy danych.

Istotnym, praktycznym atutem tej metody jest zrzucenie części zadań z użytkownika na generator. Przykładowo, *Django* jest w stanie samodzielnie obsłużyć relacje *many-to-many* pomiędzy klasami (z perspektywy bazy danych encjami). Użytkownik musi jedynie użyć odpowiedniej funkcji (w tym przypadku *ManyToManyField*) podczas definiowania pól. Tym samym, użytkownik nie ma potrzeby znać dokładnej struktury bazy danych, której używa aplikacja.

5.4 Budowanie projektu

Do automatyzacji budowania projektu wykorzystano technologię *Docker*. Na chwilę publikacji tej pracy kontener bazuje na obrazie `python:3` i w procesie budowy uwzględnia instalację narzędzi, które są potrzebne, aby działały utworzone wtyczki. Dla każdej komendy napisanej w *Dockerfile* tworzona jest oddzielna warstwa kontenera. W przypadku dodania kolejnej wtyczki i potrzeby doinstalowania narzędzia w kontenerze, jego budowa nie będzie wykonywała się od zera, a od momentu, w którym została dokonana zmiana. Takie rozwiązanie ułatwia rozwijanie projektu i oszczędza miejsce na dysku przy kolejnych modyfikacjach. Listing 5.3 prezentuje pełny skrypt *Dockerfile*.

Listing 5.3: Plik `Dockerfile` rozwiązania RedMirror

```
FROM python:3
ENV PYTHONUNBUFFERED 1

WORKDIR /root
RUN apt-get update && apt-get install -y ruby-full git unzip nmap ↙
    ↳ bsdmainutils dnsutils
RUN wget https://go.dev/dl/go1.20.3.linux-amd64.tar.gz && \
    rm -rf /usr/local/go && \
    tar -C /usr/local -xzf go*.tar.gz && \
    rm go*.linux-amd64.tar.gz
RUN gem install wpscan
RUN wget ↙
    ↳ https://github.com/drwetter/testssl.sh/archive/refs/tags/v3.0.8.zip ↙
    ↳ && \
    unzip *.zip && \
    rm *.zip && \
    mv testssl.sh-* testssl
RUN /usr/local/go/bin/go install -v ↙
    ↳ github.com/projectdiscovery/nuclei/v2/cmd/nuclei@latest

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt

RUN /root/go/bin/nuclei -update-templates

ADD . /code/
```

Projekt wymaga także bazy danych. W celu jej uruchomienia wykorzystano kolejny mechanizm *Dockera* - *docker-compose*. Buduje on opisany wcześniej obraz oraz tworzy kontener z bazą danych *PostgreSQL*. Listing 5.4 prezentuje pełny skrypt *docker-compose.yml*.

Listing 5.4: Plik `docker-compose.yml` rozwiązania RedMirror

```
version: "3.9"
services:
  db:
    image: postgres:latest
    volumes:
      - postgres_data:/var/lib/postgresql/data/
    environment:
      - "POSTGRES_HOST_AUTH_METHOD=trust"
      - "POSTGRES_NAME=postgres"
      - "POSTGRES_USER=postgres"
      - "POSTGRES_PASSWORD=postgres"
    ports:
      - "5432:5432"
    expose:
      - "5432"
  web:
    build: .
    command: bash -c "python manage.py makemigrations && python manage.py ↙
       ↳ migrate && python manage.py runserver 0.0.0.0:8001"
    volumes:
      - .:/code
    environment:
      - "POSTGRES_NAME=postgres"
      - "POSTGRES_USER=postgres"
      - "POSTGRES_PASSWORD=postgres"
      - "WPSCAN_KEY=${WPSCAN_KEY}"
    ports:
      - "8001:8001"
    depends_on:
      - db
    extra_hosts:
      - "host.docker.internal:host-gateway"

volumes:
  postgres_data:
```

Obecna konfiguracja *docker-compose* jest stworzona w celach rozwojowych aplikacji, a nie w celu uruchamiania jej w środowisku produkcyjnym. Przykładem konfiguracji, która nie powinna mieć miejsca, jest wykorzystanie `extra_hosts: - "host.docker.internal:host-gateway"`, które umożliwia łączenie się kontenera bezpośrednio z serwisami działającymi na hoście po adresie IP `127.0.0.1`. Było to wykorzystywane w przypadku, gdy uruchamiano przykładowe aplikacje do skanowania przez RedMirror. Z taką konfiguracją, przykładowa aplikacja może być uruchomiona w oddzielnym kontenerze (niepowiązanym z tymi z *docker-compose*), a mimo to będzie osiągalna dla RedMirror. Innym przykładem nieidealnej konfiguracji jest użycie poświadczeń `postgres:postgres`, które powinny być generowane losowo. Z drugiej strony takie rozwiązanie byłoby dopuszczalne, jeśli kontenery tworzone w *docker-compose* działałyby w jednej sieci (domyślne rozwiązanie w *Docker*).

5.5 Przygotowanie środowiska pod testy narzędzi

W tej sekcji opisany jest sposób konfiguracji/uruchomienia narzędzi, które były opisane w rozdziale Generyczny skaner bezpieczeństwa. Opisy dotyczą jedynie narzędzi, których uruchomienie wymagało specyficznych zmian, a nie standardowej konfiguracji.

5.5.1 Burp Suite Enterprise Edition

Narzędzie pobrano z oficjalnej strony Burp Suite [34]. Do uruchomienia Burp Suite Enterprise Edition wykorzystano narzędzie *Docker*. Listing 5.5 i Listing 5.6 pokazują plik `Dockerfile` oraz komendy służące do budowy obrazu i uruchomienia kontenera z narzędziem.

Listing 5.5: Plik `Dockerfile` obrazu pod Burp Suite Enterprise Edition.

```
FROM ubuntu
COPY burpsuite_enterprise_linux_v2023_4_1.sh ↙
    ↳ /root/burpsuite_enterprise_linux_v2023_4_1.sh
WORKDIR /root
ENTRYPOINT [ "/bin/bash", "-l", "-c" ]
```

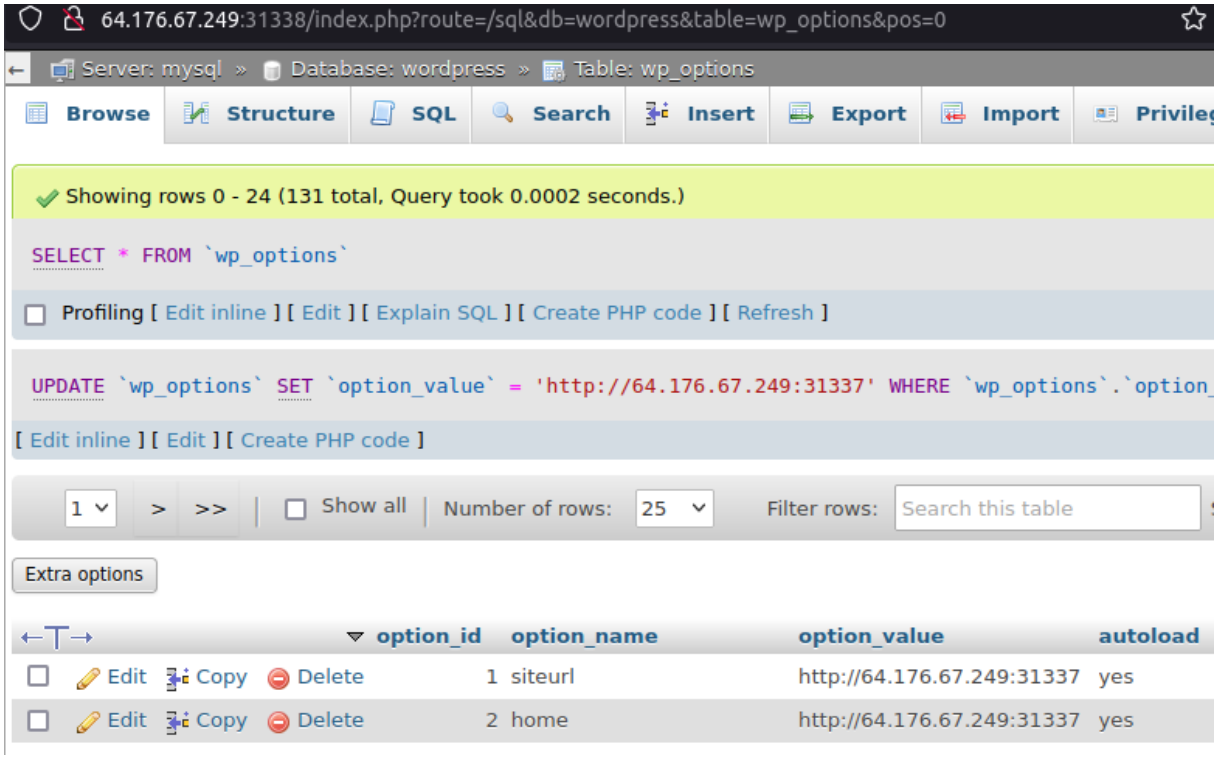
Listing 5.6: Budowa obrazu i uruchomienie kontenera pod Burp Suite

```
docker build .
# zostanie zwrocony ID stworzonego obrazu (dalej ID_OBRAZU)
docker run -it --add-host=host.docker.internal:host-gateway -p 8888:8080 ↙
    ↳ ID_OBRAZU
```

Dodatkowa konfiguracja w parametrze `--add-host` pozwala na dostęp aplikacji działających w kontenerze do portów otwartych na hoście. Porty te mogą być także portami wystawionymi przez inne kontenery działające na tej samej maszynie.

5.5.2 Damn Vulnerable WordPress

Instancja *DVWP* została uruchomiona poprzez *Docker* na zewnętrznym prywatnym serwerze wirtualnym (VPS) z systemem *Ubuntu 23.04* zgodnie z instrukcjami [80] oraz [37]. Domyślna konfiguracja nie pozwala jednak na testowanie aplikacji z zewnątrz, ponieważ przypisanym adresem strony w WordPress jest `http://127.0.0.1/`. Przez powyższe, strona w praktyce nie będzie użyteczna z zewnątrz ze względu na próby ładowania zasobów z lokalnego hosta, a nie serwera, na którym działa aplikacja. Konfiguracja została zatem zmieniona, korzystając z *phpmyadmin* uruchomionego razem z *DVWP* na porcie 31338. Korzystając z podanych w instrukcji danych zalogowano się do niego i zmieniono opcje *siteurl* oraz *home* na IP VPSa wraz z protokołem. Rysunek 5.11 prezentuje zmiany.



Rysunek 5.11: Modyfikacja rekordów w tabeli `wp_options`

6. Podsumowanie

Praca magisterska poświęcona była badaniu i realizacji koncepcji Generycznego skanera bezpieczeństwa - narzędzia mającego na celu wykrywanie podatności w aplikacjach internetowych. Rozważany system stanowi połączenie wielu mniejszych narzędzi, które w sumie umożliwiają holistyczne podejście do bezpieczeństwa aplikacji internetowych.

Zadaniem Generycznego skanera bezpieczeństwa jest przeprowadzanie kompleksowych analiz, które w dużym stopniu automatyzują proces identyfikacji podatności. Poprzez skoordynowane działanie wielu mniejszych narzędzi, system jest w stanie przeprowadzić gruntowne i wszechstronne skany, idąc daleko poza to, co możliwe byłoby przy użyciu jednego, izolowanego narzędzia.

Podsumowując, praca ta stanowi wkład do dziedziny bezpieczeństwa aplikacji internetowych, prezentując praktyczne podejście do integracji wielu narzędzi w celu przeprowadzenia dogłębnej analizy bezpieczeństwa. Implementacja Generycznego skanera bezpieczeństwa demonstruje, jak różne mniejsze narzędzia mogą być połączone w jedno skuteczne rozwiązanie, zwiększające wykrywalność potencjalnych podatności i przyczyniające się do tworzenia bezpieczniejszych aplikacji internetowych.

6.1 Problemy do zaadresowania

Ta sekcja opisuje problemy, na które należy zwrócić uwagę w najbliższym czasie. Są to działania, które mogą istotnie wpłynąć na korzystanie z aplikacji i jej dalszy rozwój.

- Należy zaimplementować *timeout* dla działania wtyczek, aby wyeliminować możliwość zawieszenia się procesu skanowania.
- Na obecną chwilę aplikacja nie posiada żadnego mechanizmu uwierzytelnienia i autoryzacji.
- Rozwiązanie polegające na podawaniu hosta lub adresu IP powinno zostać zmienione. W przypadku podania adresu IP aplikacje webowe w praktyce nie zostaną przetestowane pod kątem bezpieczeństwa, co może być zaskoczeniem dla użytkownika skanera. Dzieje się tak za sprawą konfiguracji wirtualnych hostów w serwerach webowych takich jak *Apache2* czy *nginx*. Zazwyczaj unika się sytuacji, gdy aplikacja jest dostępna z poziomu FQDN (*fully qualified domain name*) i adresu IP jednocześnie. W takiej sytuacji skaner w ogóle do takiej aplikacji nie dotrze - zidentyfikuje jedynie, że port 80/443 (typowe porty dla aplikacji webowych) są otwarte. Zamiast tego warto rozważyć dodanie wtyczki *discovery* i wprowadzenie zmian w rdzeniu mających na celu wyszukiwanie domen z rekordami DNS wskazującymi na skanowany adres IP. Następnie taka lista domen mogłaby być używana do przyszłych działań wtyczkami typu *scan*.
- Konieczna jest implementacja przestrzeni na zmienne wykorzystywane w projekcie. Wzorem implementacyjnym mogą być środowiska (*environments*) dostępne w aplikacji *Postman* [81], która służy do obsługi *REST API*. Mogą w niej występować środowiska przeznaczone dla obszarów o różnych wielkościach, np. foldery z zapytaniami, cała kolekcja czy też cały *workspace*. W podobnym schemacie w RedMirror należałoby dać możliwość ustalania zmiennych do wykorzystania w kontekście *Konfiguracji skanu*, *Projektu* lub całej aplikacji. Zmienne miałyby na celu przechowywanie danych, takich jak np. klucze *API*, do narzędzi, które tego wymagają.

6.2 Pomysły i refleksje

W trakcie tworzenia koncepcji oraz implementacji pojawiały się nowe pomysły na funkcje bądź ulepszenia projektu. Są one jednak mniej pilne niż problemy opisane w poprzednim rozdziale i wymagają dalszej analizy przed ich wdrożeniem.

- Warto dodać opcję zatrzymywania i wznawiania skanów.
- Implementacja planowania skanów np. poprzez mechanizm *cron* pozwoliłaby na jeszcze większą automatyzację narzędzia.
- Interfejs aplikacji mógłby zostać ulepszony poprzez dodanie wykresów pokazujących np. znalezione podatności na przestrzeni skanów.
- Aplikacja mogłaby wysyłać raporty o wykryciu nowych podatności np. w formie wiadomości e-mail.
- Warto rozważyć rozproszenie narzędzia, aby wykorzystać technologie chmurowe.
- W obecnej implementacji wiele działań związanych z bazą danych (tj. dodawanie, edycja i usuwanie rekordów) dzieje się w logice i widokach aplikacji. Zamiast tego należy rozważyć stworzenie klas typu DAO (*Data Access Object*) jako warstwę trwałą (*persistence layer*) w aplikacji. W obecnym stanie tylko niektóre działania są wyodrębnione do takich klas.

Bibliografia

[1] *Internet and social media users in the world 2023 | Statista*. Apr. 2023. URL: https://www.statista.com/statistics/617136/digital-population-worldwide data dostępu: 04/18/2023.

[2] *Załatw sprawy urzędowe przez internet na ePUAP - Gov.pl - Portal Gov.pl*. URL: https://www.gov.pl/web/gov/zalatwiaj-sprawy-urzedowe-przez-internet-na-epuap data dostępu: 07/24/2021.

[3] *What Do Hackers Want, Anyway? A Look at Different Cyberattack Goals – Channel Futures*. URL: https://www.channelfutures.com/strategy/what-do-hackers-want-anyway-a-look-at-different-cyberattack-goals data dostępu: 07/24/2021.

[4] M. McDonald. *Web Security for Developers*. William Pollock, 2020. ISBN: 978-1-5932-7994-3.

[5] *Access Control | OWASP Foundation*. May 2023. URL: https://owasp.org/www-community/Access_Control data dostępu: 05/07/2023.

[6] *OWASP Top 10:2021*. Mar. 2023. URL: https://owasp.org/Top10 data dostępu: 05/14/2023.

[7] *CWE - Common Weakness Enumeration*. URL: https://cwe.mitre.org data dostępu: 08/10/2023.

[8] *CWE - CWE-601: URL Redirection to Untrusted Site ('Open Redirect') (4.12)*. URL: https://cwe.mitre.org/data/definitions/601.html data dostępu: 08/10/2023.

[9] *CWE - CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') (4.12)*. URL: https://cwe.mitre.org/data/definitions/89.html data dostępu: 08/10/2023.

[10] *CWE - 2022 CWE Top 25 Most Dangerous Software Weaknesses*. May 2023. URL: https://cwe.mitre.org/top25/archive/2022/2022_cwe_top25.html data dostępu: 05/07/2023.

[11] *CVE - CVE*. URL: https://cve.mitre.org data dostępu: 08/10/2023.

[12] *Security advisories | Drupal.org*. May 2023. URL: https://www.drupal.org/security data dostępu: 05/07/2023.

[13] *Usage Statistics and Market Share of Content Management Systems, November 2020*. URL: https://w3techs.com/technologies/overview/content_management data dostępu: 11/25/2020.

[14] S. Inc. *Sucuri Security – audyt, skaner antywirusowy i zwiększenie bezpieczeństwa*. pl-PL. URL: https://pl.wordpress.org/plugins/sucuri-scanner/ data dostępu: 11/25/2020.

[15] *Sucuri - Website Threat Report 2019*. en-US. URL: https://sucuri.net/reports/2019-hacked-website-report/ data dostępu: 11/25/2020.

[16] *Configuring Automatic Background Updates*. en-US. Oct. 2018. URL: https://wordpress.org/support/article/configuring-automatic-background-updates/ data dostępu: 11/25/2020.

[17] *GDPR Archives - GDPR.eu*. May 2023. URL: https://gdpr.eu/tag/gdpr data dostępu: 05/05/2023.

[18] 3. F. G. S. El Idrissi 2 N. Berbiche and 4. M. Sbihi. *Performance Evaluation of Web Application Security Scanners for Prevention and Protection against Vulnerabilities*. URL: https://www.ripublication.com/ijaer17/ijaerv12n21_76.pdf data dostępu: 05/05/2023.

[19] *Vulnerability Scanners and Scanning Tools: What To Know | Balbix*. May 2022. URL: https://www.balbix.com/insights/what-to-know-about-vulnerability-scanning-and-tools data dostępu: 08/12/2023.

[20] D. T. Himanshu Dwivedi Chris Clark. *Mobile Application Security*. Prentice Hall, 2010. ISBN: 978-0-07-163357-4.

[21] *Dynamic Application Security Testing*. Dec. 2014. URL: https://www.techopedia.com/definition/30958/dynamic-application-security-testing-dast data dostępu: 08/12/2023.

[22] *Source Code Analysis Tools | OWASP Foundation*. Aug. 2023. URL: https://owasp.org/www-community/Source_Code_Analysis_Tools data dostępu: 08/12/2023.

[23] *What is IAST? Interactive Application Security Testing | Veracode*. Aug. 2023. URL: https://www.veracode.com/security/interactive-application-security-testing-iast data dostępu: 08/12/2023.

[24] *Runtime Application Self-Protection (RASP) | Rapid7*. [Online; accessed 12. Aug. 2023]. Aug. 2023. URL: https://www.rapid7.com/fundamentals/runtime-application-self-protection.

[25] *Software Composition Analysis: Overview and Tooling Guide*. Aug. 2023. URL: https://www.stackhawk.com/blog/software-composition-analysis-sca-overview-and-tooling-guide data dostępu: 08/12/2023.

[26] *Dynamic Application Security Testing (DAST) Software - PortSwigger*. Aug. 2023. URL: https://portswigger.net/burp/application-security-testing/dast data dostępu: 08/12/2023.

[27] *nikto*. Aug. 2023. URL: https://github.com/sullo/nikto data dostępu: 08/12/2023.

[28] *Nuclei - Community Powered Vulnerability Scanner - Index*. URL: https://nuclei.projectdiscovery.io data dostępu: 05/30/2023.

[29] *OpenVAS - Open Vulnerability Assessment Scanner*. July 2023. URL: https://openvas.org data dostępu: 08/12/2023.

[30] *Download Tenable Nessus Vulnerability Assessment*. URL: https://www.tenable.com/products/nessus data dostępu: 05/25/2023.

[31] *IT Security and Compliance Platform | Qualys, Inc.* Aug. 2023. URL: https://www.qualys.com data dostępu: 08/12/2023.

[32] *Acunetix | Web Application Security Scanner*. June 2021. URL: https://www.acunetix.com data dostępu: 08/12/2023.

[33] *What is penetration testing? | What is pen testing?* Aug. 2023. URL: https://www.cloudflare.com/learning/security/glossary/what-is-penetration-testing data dostępu: 05/14/2023.

[34] *Burp Suite Professional*. May 2023. URL: https://portswigger.net/burp/pro data dostępu: 05/18/2023.

[35] *WPScan: WordPress Security Scanner*. May 2023. URL: https://wpscan.com data dostępu: 05/18/2023.

[36] *Metasploit | Penetration Testing Software, Pen Testing Security | Metasploit*. May 2023. URL: https://www.metasploit.com data dostępu: 05/18/2023.

[37] vavkamil. *dvwp*. URL: https://github.com/vavkamil/dvwp data dostępu: 05/30/2023.

[38] *proxy - Glossary | CSRC*. URL: https://csrc.nist.gov/glossary/term/proxy data dostępu: 05/30/2023.

[39] S. Wear. *Burp Suite Cookbook: Practical recipes to help you master web penetration testing with Burp Suite*. William Pollock, 2018. ISBN: 978-1-78953-173-2.

[40] *Tenable Nessus Essentials Vulnerability Scanner*. URL: https://www.tenable.com/products/nessus/nessus-essentials data dostępu: 05/25/2023.

[41] projectdiscovery. *nuclei-docs*. URL: https://github.com/projectdiscovery/nuclei-docs data dostępu: 05/30/2023.

[42] *Usage Statistics and Market Share of Content Management Systems, May 2023*. URL: https://w3techs.com/technologies/overview/content_management data dostępu: 05/30/2023.

[43] ProjectDiscovery. "Implementing Nuclei into your GitHub CI/CD pipelines". In: *ProjectDiscovery.io | Blog* (). URL: https://blog.projectdiscovery.io/implementing-nuclei-into-your-github-ci-cd-for-scanning-live-web-applications data dostępu: 05/25/2023.

[44] *wpwatcher*. May 2023. URL: https://pypi.org/project/wpwatcher data dostępu: 05/30/2023.

[45] H. Holm, T. Sommestad, J. Almroth, and M. Persson. "A quantitative evaluation of vulnerability scanning". In: *Inf. Manag. Comput. Security* 19.4 (Oct. 2011). ISSN: 0968-5227.

[46] R. Amankwah, J. Chen, P. K. Kudjo, and D. Towey. "An empirical comparison of commercial and open-source web vulnerability scanners". In: *Software: Practice and Experience* 50.9 (July 2020). ISSN: 0038-0644.

[47] A. Lis. "Comparison and analysis of web vulnerability scanners". PhD thesis. 2019. URL: https://elib.uni-stuttgart.de/handle/11682/10634.

[48] G. Tomsho. *Guide to Networking Essentials, 6th Edition*. India: Cengage Learning, Feb. 2011. ISBN: 978-81-3150213-6.

[49] L. C. Wojciech Mazurczyk. *Cyber Reconnaissance Techniques*. [Online; accessed 10. Jun. 2023]. Mar. 2021. URL: https://cacm.acm.org/magazines/2021/3/250712-cyber-reconnaissance-techniques/fulltext.

[50] nmap. *nmap*. [Online; accessed 17. Jun. 2023]. June 2023. URL: https://github.com/nmap/nmap/blob/master/nmap-services.

[51] H. Heyman and L. Brandefelt. *A Comparison of Performance & Implementation Complexity of Multithreaded Applications in Rust, Java and C++*. 2020. URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2%5C%3A1463855&dswid=-4398.

[52] *Welcome to Python.org*. Aug. 2023. URL: https://www.python.org data dostępu: 08/12/2023.

[53] T. A. Justin Seitz. *Black Hat Python*. William Pollock, 2021. ISBN: 978-1-7185-0113-3.

[54] T. Z. Michał Jaworski. *Expert Python Programming*. Packt Publishing Ltd., 2016. ISBN: 978-1-78588-685-0.

[55] *Programming language community sizes worldwide 2022 | Statista*. [Online; accessed 1. Jul. 2023]. July 2023. URL: https://www.statista.com/statistics/1241923/worldwide-software-developer-programming-language-communities.

[56] A. Bendoraitis. *Web Development with Django Cookbook*. Packt Publishing Ltd., 2016. ISBN: 978-1-78588-677-5.

[57] L. M. Alberto Boschetti. *Python Data Science Essentials*. Packt Publishing Ltd., 2016. ISBN: 978-1-78646-213-8.

[58] S. Raschka. *Python Machine Learning*. Packt Publishing Ltd., 2015. ISBN: 978-1-78355-513-0.

[59] K. R. Srinath. "Python – The Fastest Growing Programming Language". In: *International Research Journal of Engineering and Technology (IRJET)* (Dec. 2017). ISSN: 2395-0056. URL: https://www.irjet.net/archives/V4/i12/IRJET-V4I1266.pdf.

[60] Q. Nguyen. *Mastering Concurrency in Python: Create faster programs using concurrency, asynchronous, multithreading, and parallel programming*. Packt Publishing, Nov. 2018. ISBN: 978-1-78934305-2.

[61] *The Django Book*. [Online; accessed 2. Jul. 2023]. July 2023. URL: https://buildmedia.readthedocs.org/media/pdf/djangobook/latest/djangobook.pdf.

[62] *Django*. Aug. 2023. URL: https://www.djangoproject.com data dostępu: 08/12/2023.

[63] F. Shadhin. "The MVT Design Pattern of Django - Python in Plain English". In: *Medium* (Jan. 2022). ISSN: 8476-1582. URL: https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582.

[64] *Security in Django*. [Online; accessed 2. Jul. 2023]. July 2023. URL: https://docs.djangoproject.com/en/4.2/topics/security.

[65] *Docker overview*. URL: https://docs.docker.com/get-started/overview data dostępu: 07/05/2023.

[66] *Layers*. URL: https://docs.docker.com/build/guide/layers data dostępu: 07/05/2023.

[67] B. Oggl and M. Kofler. *Docker: Practical Guide for Developers and Devops Teams (The Rheinwerk Computing)*. Rheinwerk Computing, Jan. 2023. ISBN: 978-1-49322383-1.

[68] *PostgreSQL*. URL: https://www.postgresql.org data dostępu: 07/22/2023.

[69] I. C. Education. "PostgreSQL vs. MySQL: What's the Difference? - IBM Blog". In: *IBM Blog* (). URL: https://www.ibm.com/blog/postgresql-vs-mysql-whats-the-difference data dostępu: 07/22/2023.

[70] *Stack Overflow Developer Survey 2022*. URL: https://survey.stackoverflow.co/2022/#databases data dostępu: 07/22/2023.

[71] *Introducing ChatGPT*. URL: https://openai.com/blog/chatgpt data dostępu: 07/05/2023.

[72] *About GitHub Copilot for Individuals - GitHub Docs*. URL: https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals data dostępu: 07/05/2023.

[73] *Overleaf Features & Benefits*. URL: https://www.overleaf.com/about/features-overview data dostępu: 07/06/2023.

[74] DistroWatch. *DistroWatch.com: Put the fun back into computing. Use Linux, BSD.* URL: https://distrowatch.com/search.php?pkg=Python&pkgver=3#pkgsearch data dostępu: 07/08/2023.

[75] *Free Bootstrap 5 Template - Mazer Admin Dashboard*. URL: https://zuramai.github.io/mazer data dostępu: 07/16/2023.

[76] *Key Insights from Stack Overflow's 2022 Developer Survey | Docker*. URL: https://www.docker.com/blog/key-insights-from-stack-overflows-2022-developer-survey data dostępu: 07/08/2023.

[77] *UML 2 Class Diagrams: An Agile Introduction – The Agile Modeling (AM) Method*. [Online; accessed 22. Jul. 2023]. July 2023. URL: http://agilemodeling.com/artifacts/classDiagram.htm.

[78] *Django-Styleguide*. URL: https://github.com/HackSoftware/Django-Styleguide data dostępu: 07/16/2023.

[79] *3. Data model*. URL: https://docs.python.org/3/reference/datamodel.html data dostępu: 07/16/2023.

[80] *Install Docker Engine on Ubuntu*. URL: https://docs.docker.com/engine/install/ubuntu data dostępu: 05/30/2023.

[81] *Managing environments | Postman Learning Center*. URL: https://learning.postman.com/docs/sending-requests/managing-environments data dostępu: 07/27/2023.

Lista rysunków

Lista listingów