Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Elektrotechniki Teoretycznej i Systemów Informacyjno-Pomiarowych

Praca dyplomowa magisterska

na kierunku Informatyka Stosowana

w specjalności Cyberbezpieczeństwo

Porównanie dynamicznego testowania bezpieczeństwa aplikacji z testami statycznymi pod kątem wybranych rodzajów luk bezpieczeństwa

inż. Hubert Czarnowski

numer albumu 316986

promotor

dr inż. Bartosz Chaber

WARSZAWA 2024

Porównanie dynamicznego testowania bezpieczeństwa aplikacji z testami statycznymi pod kątem wybranych rodzajów luk bezpieczeństwa

Streszczenie

To jest streszczenie. To jest trochę za krótkie, jako że powinno zająć całą stronę.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Słowa kluczowe: A, B, C

Comparison of dynamic application security testing with static analysis security testing based on a selected security vulnerabilities

Abstract

This is abstract. This one is a little too short as it should occupy the whole page.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Keywords: X, Y, Z

Spis treści

Rozdział 1

Wstęp

W dzisiejszym świecie aplikacje internetowe są kluczowym elementem technologicznym zyskującym popularność każdego roku. Są one wykorzystywane jako źródło rozrywki, środek do dokonywania zakupów, bankowość internetowa, a także stanowią integralną część aplikacji mobilnych. Wraz z rosnącą popularnością aplikacji internetowych, zapewnienie odpowiedniego poziomu bezpieczeństwa dla każdej z nich staje się kluczowe [3].

Regularna aktualizacja i bezpieczne tworzenie aplikacji internetowych są niezwykle istotne, zgodnie z zaleceniami m.in. organizacji Open Web Application Security Project (OWASP) [15]. Należy zauważyć, że atakujący czasami mogą jedynie potrzebować znalezienia i wykorzystania pojedynczej podatności, aby zagrozić integralności, poufności i dostępności danych w aplikacji internetowej.

W poniższych podrozdziałach zostały omówione aspekty dbania o bezpieczeństwo związane z testami penetracyjnymi, automatyzacją testów, a także charakterystyką narzędzi pozwalające na automatyzację.

1.1 Testy penetracyjne

Jedną z metod oceny bezpieczeństwa aplikacji internetowej są testy penetracyjne. Jest to procesem weryfikującym bezpieczeństwo systemów informatycznych polegająca na przeprowadzeniu kontrolowanego ataku na np.: aplikację internetową zgodnie z obowiązujący prawem. Głównym celem testów jest sprawdzenie, czy możliwe jest złamanie zabezczpień testowanej aplikacji oraz zidentyfikowanie potencjalnych zagrożeń związanych z zagrożeniami cyberbezpieczeństwa. Należy jednak pamiętać, że testy penetracyjne nie zapewniają pełnej gwarancji znalezienia wszystkich niebezpiecznych luk w zabezpieczeniach aplikacji. Stanowią one raczej jeden ze sposobów dbania o cyber bezpieczeństwo w organizacjach, mając na celu zminimalizowanie ryzyka związanego z różnego rodzaju incydentami dotyczących: m.in bezpieczeństwa danych oraz ciągłością działania aplikacji. Testy penetracyjne koncentrują się nie tylko na znajdowaniu słabości w zabezpieczeniach, lecz także na badaniu procesów biznesowych pod względem bezpieczeństwa danych. Takie podejście pozwala na zapewnienie kompleksowej oceny bezpieczeństwa, uwzględniającej nie tylko aspekty techniczne,

ale również organizacyjne. Zrozumienie procesów biznesowych jest kluczowe, aby dostosować środki zabezpieczające do specyficznych wymagań i charakteru danej organizacji.

Najpopularniejszymi metodami testów penetracyjnych są: - testy czarnej skrzynki (ang. black box), które polegają na przeprowadzeniu oceny bezpieczeństwa z minimalną wiedzą na temat danego systemu lub organizacji, - testy białej skrzynki (ang. white box), charakteryzują się pełnym dostępem do testującej aplikacji, skupiające się m.in. na analizie kodu oprogramowania. - testy szarej szkynki (grey box), łączą obie powyższe metody testów czarnej i białej skrzynki, co pozwala na bardziej wszechstronną analizę bezpieczeństwa.

1.2 Automatyzacja testów penetracyjnych

Automatyzacja testów penetracyjnych stała się ważnym elementem w zapewnianiu bezpieczeństwa w procesie wytwarzania oprogramowania. Idealnym scenariuszem jest przeprowadzanie testów penetracyjnych zawsze po aktualizacji aplikacji internetowej. Niestety, testy wykonane przez zespół specjalistów ds. bezpieczeństwa wiążą się z większymi kosztami dla organizacji, dlatego takie testy powinny być przeprowadzane tylko po zakończeniu prac nad istotnymi funkcjonalnościami systemu [1]. Jednakże pojawia się problem mniejszych aktualizacji systemu, które również wymagają weryfikacji pod względem bezpieczeństwa.

W obliczu tego wyzwania, automatyzacja testów penetracyjnych oferuje rozwiązanie, pozwalając na ciągłe i efektywne sprawdzanie systemów w poszukiwaniu błędów związanych z bezpieczeństwem. Wprowadzenie automatyzacji do procesów testów penetracyjnych wymaga jednak zastosowania odpowiednich narzędzi i technologii. Te narzędzia, często zintegrowane z systemami ciągłej integracji i ciągłego wdrażania (CI/CD), pozwalają na bezproblemowe wprowadzanie bezpieczeństwa w całym cyklu życia oprogramowania [17]. Przykłady takich narzędzi obejmują zautomatyzowane skanery bezpieczeństwa i systemy do zarządzania lukami w zabezpieczeniach, które umożliwiają szybką i skuteczną analizę potencjalnych zagrożeń.

1.3 Charakterystyka wybranych narzędzi wykorzystywanych do automatyzacji testów penetracyjnych

W kontekście automatyzacji testów penetracyjnych,DAST (Dynamic Application Security Testing) oraz SAST (Static Application Security Testing) są uznawane za najbardziej popularne narzędzia. Stanowią one integralną część strategii bezpieczeństwa, odpowiednio umożliwiając przeprowadzenie analizy kodu pod kątem zagrożeń oraz automatycznych testów bezpiczeństwa.

W poniższych podrozdziałach zostały opisane typy tych narzędzi.

1.3.1 Dynamiczne Teestowanie Bezpieczeństwa Aplikacji (DAST)

Narzędzia do Dynamicznego Testowania Bezpieczeństwa Aplikacji (DAST) są zaprojektowane do oceny bezpieczeństwa aplikacji w czasie rzeczywistym, poprzez analizowanie ich zachowania w warunkach produkcyjnych. Działanie tych narzędzi kwalifikuje się jako testy penetracyjne typu czarnej skrzynki, które polegają na automatycznym skanowaniu aplikacji internetowych bez dostępu do wewnętrznej struktury kodu źródłowego. Charakterystyczną cechą DAST jest przeprowadzanie testów z ograniczoną ilością informacji o aplikacji, co zazwyczaj ogranicza się do adresu URL. Takie testy mają na celu zidentyfikowanie potencjalnych luk i braków w zabezpieczeniach, symulując działania potencjalnego atakującego, który stara się włamać do aplikacji z poziomu użytkownika. Dzięki temu narzędzia DAST oferują wszechstronną ocenę bezpieczeństwa aplikacji w warunkach zbliżonych do rzeczywistego użytkowania. [20].

1.3.2 Statyczne Testowanie Bezpieczeństwa Aplikacji (SAST)

Narzędzia typu SAST, czyli Statyczne Testowanie Bezpieczeństwa Aplikacji, odgrywają kluczową rolę już na wczesnym etapie programowania, umożliwiając szczegółową analizę kodu źródłowego aplikacji internetowej w poszukiwaniu potencjalnych podatności bezpieczeństwa, błędów programowania i innych zagrożeń. W przeciwieństwie do testów typu czarnej skrzynki, gdzie dostęp do wewnętrznej struktury aplikacji jest ograniczony, narzędzie typu SAST umożliwia kompleksową analizę struktury kodu, identyfikując potencjalne ryzyka na etapie rozwoju aplikacji [21].

Ta metoda testowania, klasyfikowana jako forma testowania penetracyjnego typu białej skrzynki, charakteryzuje się pełnym dostępem do kodu źródłowego aplikacji [16]. Wykonanie takich testów dostarcza bardziej szczegółowych informacji o bezpieczeństwie aplikacji, między innymi analizując funkcjonalności niedostępne z poziomu użytkownika. Wyniki również mogą być wykorzystane do wdrażania skutecznych praktyk bezpieczeństwa na poziomie tworzenia kodu źródłowego, zapewniając wysoki poziom ochrony przed wdrożeniem aplikacji do środowiska produkcyjnego.

Rozdział 2

Analiza wybranych podatności w aplikacjach internetowych

W tym rozdziale zostały przedstawione trzy kluczowe podatności, które stanowią zagrożenie dla aplikacji internetowych: Wstrzyknięcia NoSQL (ang. NoSQL Injection), zanieczyszczenia prototypu (Prototype Pollution) oraz wyłudzenie pamięci podręcznej sieciowej (Web Cache Deception). Zrozumienie tych potencjalnych słabych punktów ma znaczenie zarówno w ramach testów penetracyjnych, jak i w procesach produkcji oprogramowania.

2.1 Analiza podatnosci Prototype pollution

Podatność związana z zanieczyszczeniem prototypu (Prototype Pollution) występuje tylko w środowiskach związanych z językiem programowania JavaScript, takich jak np.: Node.js, który jest używany do tworzenia serwerowych aplikacji internetowych. W języku JavaScript obiekt pełni rolę podstawowego szablonu dla wszystkich nowo utworzonych obiektów, co oznacza, że nawet pusty obiekt posiada pewne dziedziczone właściwości. Możliwość redefiniowania obiektów np.: przy wykorzystywaniu klonowania obiektów otwiera drzwi do potencjalnej podatności zanieczyszczenia prototypu, ponieważ wszystkie obiekty dziedziczące od głównego obiektu mogą zyskiwać nowe metody [12] [13].

Poniższy Listing 1 przedstawia zachowanie obiektów w języku JavaScript.

```
1 Object.prototype.foo = function() {
2 console.log("New function foo");};
3 const obj = {};
4 obj.foo(); // Displays "New function foo"
```

Listing 1. Przykład zmiany prototypów w Javascript

Z tego powodu, podczas projektowania i programowania aplikacji w języku JavaScript, kluczowe jest, aby deweloperzy świadomie zarządzali prototypami i starali się zminimalizować ryzyka związane z zanieczyszczeniem prototypu, zapewniając bezpieczeństwo kodu, szczególnie w kontekście manipulacji

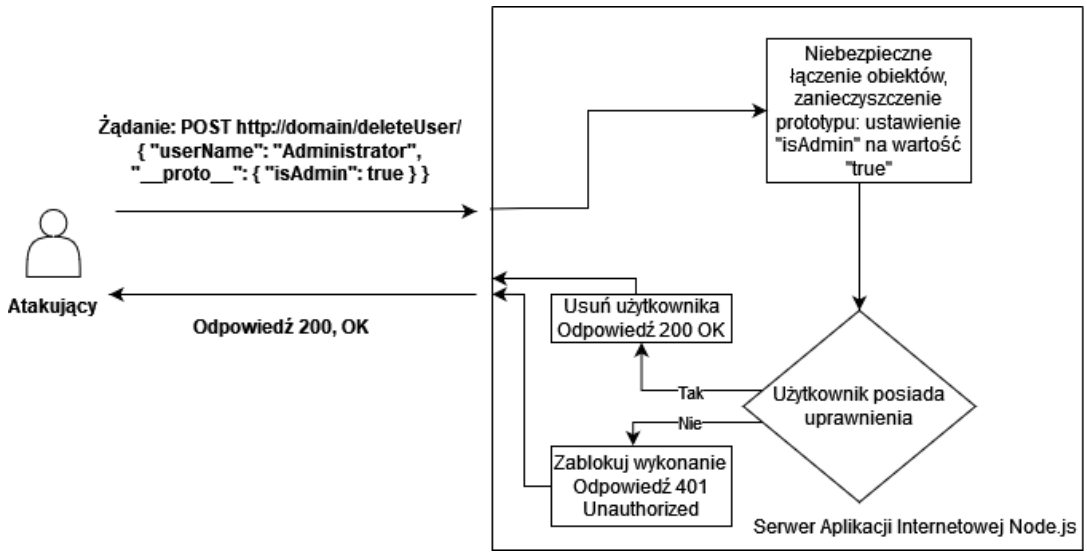
obiektami. Pierwsze publikacje wskazujące na to zjawisko zostały przedstawione przez Arteau [4], który zidentyfikował popularne biblioteki Node.js dotknięte tym problemem. Atakujący dążą do zidentyfikowania interfejsów API, które akceptują parametry. Jeśli aplikacja po stronie serwera niepoprawnie obsługuje operacje takie jak klonowanie obiektów, istnieje potencjalne ryzyko ataku.

Poniżej w Listingu 2 został przedstawiony przykład, w jaki sposób atakujący może próbować manipulować parametrami wejściowymi w formacie JSON[1] do aplikacji, wykorzystując potencjalne podatności w obsłudze tych parametrów przez serwer. Przykład wskazuje na próbę dodania nowego parametru `isAdmin` o wartości `true`, w celu obejścia prostych walidacji uprawnień użytkownika.

```
1 { "__proto__": { "isAdmin": true } }
```

Listing 2. Przykład zmiany prototypów w JSON

Rysunek 1 przedstawia przykładowy schemat ataku z wykorzystaniem podatności typu prototype pollution. Atakujący wysyła żądanie POST do serwera aplikacji z zamiarem usunięcia konta użytkownika, gdzie taka operacja jest zazwyczaj zastrzeżona dla kont o wyższych uprawnieniach, takich jak administrator. Żądanie zawiera dane w formacie JSON z nazwą użytkownika ustawioną na "Administrator" oraz z złośliwym kodem, którego celem jest zanieczyszczenie globalnego prototypu obiektu JavaScript, poprzez ustawienie właściwości `"isAdmin"` na wartość `"true"`. W przypadku złej implementacji mechanizmu obsługującego obiekty, tak skonstruowane żądanie spowoduje dodanie złośliwego prototypu, umożliwiając nieuprawnionemu użytkownikowi wykonanie operacji usunięcia konta. Natomiast w przypadku poprawnego działania aplikacji, taka operacja zakończy się zwróceniem błędu z statusem HTTP 401, oznajmiającym brak uprawnień do wykonania żądanej operacji.



Rysunek 1. Schemat ataku z wykorzystaniem prototype pollution na aplikację korzystającej z serwera Node.js

[1] JavaScript Object Notation

2.2 Analiza podatnosci Nosql Injection

NoSQL (Not Only SQL) to rodzina systemów zarządzania bazami danych, która różni się od tradycyjnych baz danych SQL pod względem metod przechowywania i organizacji danych. Wraz ze wzrostem popularności systemów NoSQL, takich jak MongoDB, Redis, Cassandra, wzrasta również ryzyko ataków związanych z bezpieczeństwem. Jednym z takich zagrożeń jest wstrzyknięcie kodu NoSQL (ang. NoSql Injection), podobna do tradycyjnego ataku wstrzykniecia kodu SQL, ale ukierunkowana na bazy danych, które nie wykorzystują relacyjnych struktur.

Wstrzyknięcia NoSQL jest rodzajem ataku, który wykorzystuje luki w zabezpieczeniach aplikacji korzystającej z baz danych NoSQL. W przeciwieństwie do tradycyjnych baz danych opartych na SQL, NoSQL opiera się na różnych modelach danych, takich jak dokumenty, kolumny czy grafy. Atakujący, wykorzystując braki w walidacji danych wejściowych, są w stanie manipulować zapytaniami do bazy danych przez wstrzykiwanie złośliwych fragmentów zapytań. Konsekwencje takich działań mogą obejmować ujawnienie wrażliwych informacji, modyfikację danych w bazie danych, przeciążenie usługi lub obejście procesów uwierzytelnienia i autoryzacji w aplikacji [6].

Najczęstszym powodem, dla którego aplikacja internetowa jest podatna na ataki, jest brak walidacji danych wejściowych lub niewłaściwe filtrowanie. Dane te są zwykle przesyłane za pomocą formularzy online, które komunikują się bezpośrednio z bazą danych, na przykład gdy użytkownik przesyła dane do aplikacji w celu uzupełniania informacji o profilu. Poniżej w listingu 3, przedstawiono przykładowy kod aplikacji który jest podatny na ataki wstrzyknięcia NoSQL atakami z uwzględnięciem bazy danych MongoDB, będącej jedną z najpopularniejszych baz danych NoSQL [2].

```
1 db.users.find({
2 username: req.body.username,
3 password: req.body.password)};
```

Listing 3. Przykład niewystarczającej walidacji danych wejściowych w JavaScript podczas interakcji z MongoDB

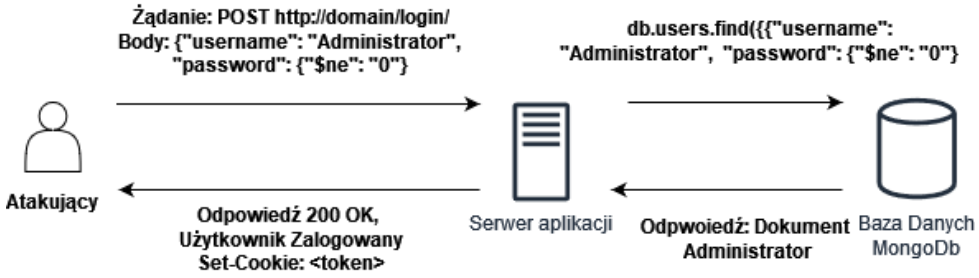
Powyższy przykład ilustruje przesyłanie danych z parametrów zapytania do aplikacji w niewłaściwy sposób. Poprzez użycie odpowiednich operatorów danych w bazach danych NoSQL, takich jak `$where, $ne, $ge, $regex`, atakujący są w stanie naruszyć bezpieczeństwo danych w aplikacji internetowej. Listng 4 przedstawia potencjalne przesłanie danych w fromacie JSON, z wykorzystaniem operatorów dancyh, które wykonają atak NoSql Injeciton.

```
1 {"username": {"$gt": ""},
2  "password": {"$ne": "0"}}
```

Listing 4. Przykład wysłania danych wejściowych, które wywołają atak polegający na wstrzyknięciu NoSql pozwalający ominąć proces uwierzytelniania

Rysunek 2 prezentuje przykładowy proces wykonania ataku typu NoSQL Injection na aplikację wykorzystującą bazę danych MongoDB. Atak rozpoczyna się od wysłania żądania POST na punkt końcowy API (ang. endpoint) `/login` serwera aplikacji, zawierającego dane w formacie JSON do

uwierzytelnienia użytkownika. Dane sugerują, że atakujący próbuje zalogować się na użytkownika Administrator, używając hasła, które zawiera złośliwy kod z operatorem `"$ne"` ustawionym na `"0"`, oznaczającym "różne od zera", w celu zmanipulowania logiki zapytania do bazy danych. Taka operacja będzie zawsze zwracać poprawny wynik, bez potrzeby znajomości prawdziwego hasła użytkownika. Brak walidacji po stronie serwera przekazuje dane bezpośrednio do bazy danych, otrzymując pozytywną odpowiedź. W rezultacie, atakujący otrzymuje odpowiedź ze statusem 200 OK i token sesji, co sugeruje nieuprawnione nadanie mu uprawnień użytkownika.



Rysunek 2. Schemat ataku NoSql Injection na aplikację korzystającej z MongoDb

2.3 Analiza podatności Web Cache Deception

Atak metodą Web Cache Deception (WCD) wykorzystuje luki w implementacji lub konfiguracji pamięci podręcznej serwera, umożliwiając nieuprawniony dostęp do zasobów, które powinny być chronione. W ramach tego ataku napastnik jest w stanie manipulować mechanizmami cache'owania, co może prowadzić do nieautoryzowanego dostępu do wrażliwych danych, które w normalnych warunkach byłyby zabezpieczone [14]. Atak ten może zatem omijać mechanizmy kontroli dostępu i kompromitować poufne informacje.

Pierwsze zidentyfikowanie tej podatności zostało zaprezentowane przez O. Gila w 2017 roku [7], gdzie opisano rzeczywisty przypadek wykorzystania tej podatności w systemie płatności online, co umożliwiło dostęp do poufnych danych prywatnego konta płatniczego. To wykorzystanie ułatwiło dostęp do poufnych danych związanych z prywatnym kontem płatniczym.

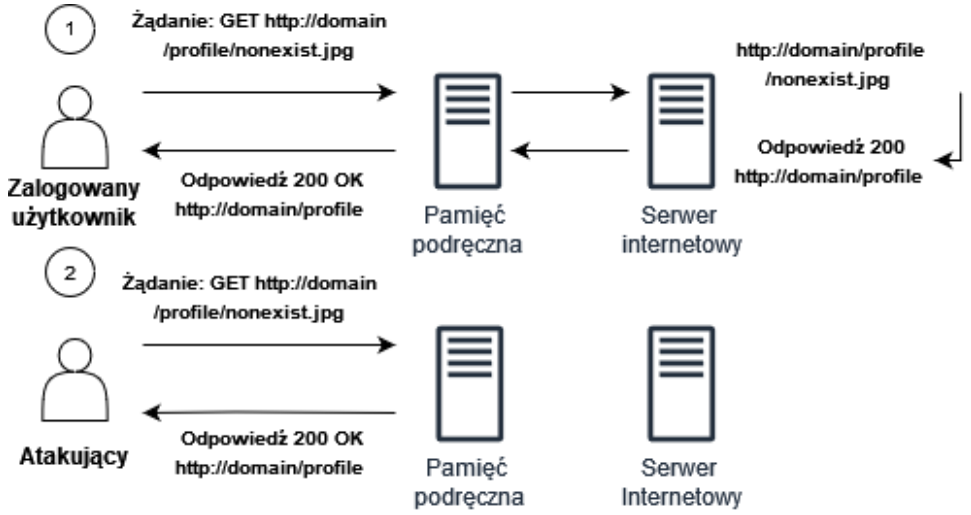
Poniżej w listngu 5 przedstawiono przykład adresu URL wykorzystywanego w ataku typu Web Cache Deception, gdzie żądanie HTTP odwołuje się do nieistniejącego pliku graficznego.

```
1 http://domain/profile/nonexist.jpg.
```

Listing 5. Przykład ataku metodą Web Cache Deception (WCD) poprzez żądanie nieistniejącego pliku.

Na rysunku 3 przedstawiono dwa scenariusze zachodzące po sobie w kontekście ataku typu Web Cache Deception. W pierwszym przypadku zalogowany użytkownik próbuje uzyskać dostęp do nieistniejącego pliku `nonexist.jpg` w katalogu `/profile`. Mimo że plik nie istnieje, serwer aplikacji zwraca status HTTP 200 OK, co może wskazywać na zapisanie się profilu użytkownika w

pamięci podręcznej serwera. W drugim scenariuszu atakujący wykonuje to samo żądanie. Ponieważ odpowiedź na to żądanie została już zapisana w pamięci podręcznej, atakujący również otrzymuje odpowiedź 200 OK z zawartością profilu użytkownika zalogowanego. Takie wykorzystanie podatności pozwala atakującemu uzyskać dostęp do danych, które powinny pozostać prywatne.



Rysunek 3. Schemat ataku Web Cache Deception Injection na aplikację internetową

Rozdział 3

Badanie narzędzi do automatycznych wykonywania testów penetracyjnych

W poniższym rozdziale przeprowadzono i scharakteryzowano wyniki porównania narzędzi do automatycznych testów penetracyjnych. Badanie obejmowało dwie serie testów, gdzie w pierwszej serii zaprezentowano szerszy zakres narzędzi, które zostały uruchomione w wersji domyślnej. W drugiej serii wybrane narzędzia zostały wzbogacone o dodatkowe rozszerzenia, a następnie poddane porównaniu. Celem danego badanie jest zrozumienie efektywności i skuteczności w wykrywaniu podatności oraz przedstawienie . W ramach badania została stworzono autorską aplikację internetową, zawierającą takie podatności jak: NoSQL Injection, Prototype Pollution oraz Web Cache Deception.

3.1 Środowisko testowe

Badanie skuteczności narzędzi do automatycznych testów penentracyjnych w ramach niniejszej analizy obejmuje wykorzystanie narzędzi typu DAST i SAST. Testy zostały przeprowadzone w wykorzystaniu systemu Kali Linux 2023.4 z wersją jądra Linux 6.5.0. Poniżej zostały przedstawione kluczowe narzędzia użyte w analizie.

Wybrane narzędzia typu DAST:

- ZAP (Zed Attack Proxy): Jest to narzędzie z otwartym kodem źródłowym, które umożliwia skanowanie aplikacji w poszukiwaniu podatności bezpieczeństwa. ZAP oferuje zarówno automatyczne testowanie, jak i zautomatyzowane ataki aplikacyjne, co pomaga w wykrywaniu potencjalnych zagrożeń. Często stosowane w środowiskach deweloperskich, narzędzie to pozwala na elastyczne dostosowywanie testów do specyficznych potrzeb danego projektu.
- Burp Suite Professional: Burp Suite to wszechstronne narzędzie do testowania bezpieczeństwa aplikacji webowych, obejmujące szeroką gamę funkcji. Zapewnia ono możliwość skanowania pod kątem podatności oraz ręcznego testowania, co pozwala analitykom bezpieczeństwa na dokładną analizę aplikacji. Burp Suite wyróżnia się intuicyjnym interfejsem użytkownika oraz dokładnością w raportowaniu wyników testów. Ponadto, narzędzie to jest wyposażone w liczne

dodatki, które zwiększają jego funkcjonalność i dostosowują je do rozmaitych wymagań analizy bezpieczeństwa.

Wybrane narzędzia typu SAST:

- SemGrep: Jest to narzędzie open-source przeznaczone do statycznej analizy kodu. SemGrep skupia się na identyfikacji potencjalnych błędów i luk bezpieczeństwa, a także na sprawdzaniu zgodności z przyjętymi standardami kodowania. Charakteryzuje się wykorzystaniem reguł opartych na wyrażeniach regularnych i wzorcach, co umożliwia użytkownikom tworzenie spersonalizowanych kryteriów analizy kodu.
- SonarQube: SonarQube to kompleksowa platforma automatyzująca proces oceny jakości kodu. Zapewnia ona dogłębną statyczną analizę kodu, mierzy pokrycie testami i identyfikuje błędy w kodzie, które mogą stanowić potencjalne zagrożenia bezpieczeństwa. Narzędzie to jest kluczowe w procesie wykrywania i rozwiązywania problemów związanych z jakością i bezpieczeństwem kodu.

3.2 Charakterystyka aplikacji demo

W tym podrozdziale szczegółowo omówiona została demonstracyjna autorska aplikacja webowa, zaprojektowana i zaimplementowana w języku JavaScript, działająca na platformie Node.js. Aplikacja ta została specjalnie stworzona do celów badawczych i zawiera szereg świadomie zaimplementowanych podatności bezpieczeństwa, które odzwierciedlają typowe problemy napotykane w rzeczywistych środowiskach aplikacji webowych.

Podatność typu Prototype Pollution została zaimplementowana w aplikacji w trzech różnych wariantach. Pierwszy z nich wykorzystuje podatną bibliotekę Lodash w wersji 4.5.0, która jest jednym z popularniejszych narzędzi w środowisku JavaScript. Szczególnie funkcje takie jak set i merge w tej wersji biblioteki mogą prowadzić do zanieczyszczenia globalnego obiektu prototypu, co daje atakującemu możliwość manipulowania obiektami w aplikacji. Drugim elementem, w którym zastosowano podatność, jest autorskie rozwiązanie służące do scalania obiektów. Dane implementacje podatności znajdują się w funkcjach takich jak: dodawanie notatek i odczytywanie plików. W funkcji dodawania notatek atakujący może wykorzystać słabości w obsłudze obiektów do wstrzykiwania złośliwego kodu, podczas gdy w funkcji odczytywania plików podatność ta może być użyta do uzyskania nieautoryzowanego dostępu do plików lub przechowywanych danych na serwerze.

Podatność typu NoSQL Injection została zaimplementowana przy wykorzystaniu biblioteki Mongoose w wersji 5.10.14, która ułatwia komunikację z bazą danych MongoDB. Podatności te zostały wdrożone w kilku kluczowych funkcjonalnościach aplikacji, odzwierciedlając typowe problemy występujące w aplikacjach internetowych. Podatność ta została wprowadzona w trzech różnych komponentach, co opisano poniżej:

- Logowanie: Podatność umożliwia obejście mechanizmu uwierzytelniania, co prowadzi do nieautoryzowanego dostępu do aplikacji.

- Wyszukiwarka: W module wyszukiwania zaimplementowano podatność, która stwarza ryzyko nieuprawnionego dostępu do danych lub ich manipulacji poprzez złośliwe zapytania.
- System Śledzący Działania Użytkownika (ang. Tracker), który rejestruje informacje takie jak User-Agent, co pozwala serwerom na identyfikację urządzeń i przeglądarek z których korzystają użytkownicy. Wykorzystanie zaimplementowanej podatności pozwala na odczyt zbieranych danych lub ich modyfikację.

Podatność typu Web Cache Deception została zaimplementowana w mechanizmie do zapisywania danych w pamięci podręcznej serwera. W tym celu wykorzystano bibliotekę memory-cache w wersji 0.2.0, która wspomaga obsługę pamięci podręcznej. Dzięki temu aplikacja jest w stanie tymczasowo przechowywać dane, w celu przyspieszenia dostępu dla użytkowników. W tym przypadku celowo wprowadzono lukę, która pozwala na zapisywanie wrażliwych danych zawartych w profilu użytkownika.

- można zobrazować z pseudo kodem w jaki sposób są podatne aplikacje (screeny z burpa)

3.3 Metodyka Badań

W niniejszym rozdziale przedstawiono metodykę badań, która ma na celu analizę oraz porównanie narzędzi do automatycznych testów penetracyjnych aplikacji internetowych. Wyniki opierają się informacjach o narzędziach oraz kompletnych skanach przeprowadzonych na demonstracyjną aplikację internetową. W pierwszej kolejności skupiono się na badaniach porównujących parametry takie jak :

- Typ oprogramowania,
- Typ interfejsu użytkownika,
- Czas trwania skanu,
- Liczba wysłanych zapytań HTTP lub wykorzystanych regół w zależności od typu narzędzia,
- Liczba wykrytych alertów bezpieczeństwa,
- Dostępność rozszerzeń.

Następnym krokiem w procesie badań było przeprowadzenie analizy porównawczej, skupiającej się na ocenie skuteczności narzędzi w wykrywaniu podatności w ramach testowanej aplikacji internetowej.

Ostatnim etapem badań było wykonanie testów z wykorzystaniem zmodyfikowanych wersji narzędzi, które zostały uruchomione w konfiguracji niestandardowej, odbiegającej od ustawień domyślnych.

3.4 Wyniki i analiza badań

Poniższy podrozdział przedstawia wyniki i porównanie kluczowych wskaźników wydajności narzędzi skanujących.

3.4.1 Porównanie efektywności narzędzi typu SAST i DAST

Wyniki przedstawione w TAbeli I oraz Tabeli III przedstawiają rezultaty przeprowadzonych testów narzędzi w ich domyślnych konfiguracjach.Przeprowadzona analiza miała na celu zbadanie i ocenę wydajności wybranych narzędzi.

Parametry\Narzędzie DAST	**ZAP**	**BURP SUITE**
Typ oprogramowania	Otwarty kod źródłowy	Zamknięty kod źródłowy
Typ interfejsu użytkownika	Graficzny interfejs	Graficzny interfejs
Czas trwania skanu	5:01min	2:21min
Liczba wysłanych zapytań HTTP	2932	3021
Wykryte alerty bezpieczeństwa	13	8
Rozszerzenia z Marketplace	Dostępne	Dostępne

Tabela 1. Porównanie parametrów skanowania narzędzi typu DAST

Analizując dane przedstawione w Tabeli 1, można zaobserwować, że oba programu posiadają graficzny interfejs użytkownika oraz oferują dostęp do potencjalnych rozszerzeń. W zakresie różnic, Burp Suite wyróżnia się krótszym czasem skanowania oraz większa ilością wysłanych żądań do aplikacji internetowej. Ponadto narządzie ZAP wykryło większą ilość alertów bezpieczeństwa, co może sugerować, że ZAP ma wyższą wrażliwość na potencjalne zagrożenia.

Poniżej w Tabeli 2 przedstawiono skuteczność narzędzi w wykrywaniu zaimplementowanych podatności w aplikacji testowej.

Podatność\Narzędzie	**ZAP**	**BURP SUITE**
Prototype pollution 1	Niewykryta	Niewykryta
Prototype pollution 2	Niewykryta	Niewykryta
Prototype pollution 3	Niewykryta	Niewykryta
NoSql injection 1	Niewykryta	Niewykryta
NoSql injection 2	Niewykryta	Niewykryta
NoSql injection 3	Niewykryta	Niewykryta
Web Cache Deception	Niewykryta	Niewykryta
Rezultat	0/7	0/7

Tabela 2. Porównanie wyników skuteczności narzędzi typu DAST

Wyniki przedstawione w Tabeli 2 wskazują, że zastosowane narzędzia nie wykryły żadnych podatności. Może to być wynikiem konstrukcji testowanej aplikacji oraz braku obsługi definicji reguł w narzędziach, które pozwalałyby na wykrycie mniej powszechnych podatności, które niekoniecznie są związane z mniejszym ryzykiem systemu.

Kolejne badania, przedstawione w Tabeli 3 i 4, miały na celu porównanie narzędzi do Statycznego Testowania Aplikacji. Narzędzia te umożliwiają przeprowadzenie skanu kodu źródłowego napisanego dla aplikacji demonstracyjnej.

Parameters\SAST Tool	SemGrep	SonarQube
Typ oprogramowania	Otwarty kod źródłowy	Otwarty kod źródłowy
Typ interfejsu użytkownika	Wiersz poleceń and Graficzny interfejs	Wiersz poleceń and Graficzny interfejs
Czas trwania skanu	26s	10s
Ilość wykorzystanych reguł	304	264
Wykryte alerty bezpieczeństwa	13	8
Rozszerzenia z Marketplace	Dostępne	Dostępne

Tabela 3. COMPARISON OF SAST TOOL RESULTS

Wstępna analiza narzędzi SemGrep i SonarQube, które służą do statycznej analizy kodu aplikacji internetowych, dostarcza informacji na temat ich wydajności w procesie skanowania kodu. Oba narzędzia są typu otwartego kodu źródłowego (ang. open-source) i mają dostęp do sklepu z rozszerzeniami. Takie podejście twórców umożliwia wsparcie narzędzia przez społeczność, między innymi poprzez tworzenie nowych rozszerzeń do narzędzia. SemGrep charakteryzuje się dłuższym czasem skanowania w porównaniu do SonarQube, co może wynikać z większej liczby uruchamianych reguł skanujących. Te reguły również pozwalają na szerszy zakres i większą głębię analizy statycznej kodu, co jest widoczne również w zgłaszanych alertach bezpieczeństwa.

Poniżej w Tabeli 4 przedsatwiono porównanie skuteczności narzędzi w wykrywaniu podatności w kodzie aplikacji.

Vulnerability\SAST Tool	SemGrep	SonarQube
Prototype pollution 1	Niewykryta	Niewykryta
Prototype pollution 2	Niewykryta	Niewykryta
Prototype pollution 3	Niewykryta	Niewykryta
NoSql injection 1	**Wykryta**	Niewykryta
NoSql injection 2	**Wykryta**	Niewykryta
NoSql injection 3	Niewykryta	Niewykryta
Web cache deception	Niewykryta	Niewykryta
Rezultat	2/7	0/7

Tabela 4. Porównanie wyników skuteczności skanów narzędzi typu SAST

W powyższym porównaniu można zauważyć, że narzędzie Semgrep zidentyfikowało dwie podatności typu NoSql Injection, natomiast SonarQube nie zdołał wykryć żadnej z zaimplementowanych luk w kodzie aplikacji. Może być to spowodowane brakiem odpowiednich reguł skanowania mniej popularnych podatności w SonaQube.

3.4.2 Dostosowanie i usprawnienie skanów narzędzi: BurpSuite i SemGrep

W tej części rozdziału zostało opisane sposób rzoszerzenia narzędzi w celu poprawy skuteczności skanowania. Porównanie obejmowało wybrane parametry oraz skuteczność wykrywania podatności

w aplikacji internetowej. Do tego procesu wybrano dwa narzędzia które oferują szeroką i różnorodną ilość rozszerzeń przeznaczonych do wykrywania konkretnych podatności w konkretnych językach programowania.

W pierwszej kolejności ulepszenia zostały wdrożone do narzędzia Burp Suite. Twórcy oprogramowania udostępniają publiczne API, które umożliwia tworzenie nowych rozszerzeń zarówno przez twórców oprogramowania, jak i niezależnych programistów. Wykorzystano tę możliwość do znacznej rozbudowy i wzbogacenia funkcjonalności programu Burp Suite. Dodane rozszerzenia skupiają się na ulepszeniu funkcji skanujących, oferując bardziej obszerny zakres scenariuszy testowych i lepszą zdolność do wykrywania nowych typów podatności. Takie rozwiązanie znacząco zwiększa efektywność narzędzia w identyfikacji słabych punktów bezpieczeństwa. W ramach tego procesu zainstalowano następujące dodatki:

- **Retire.js** (Autor: Philippe Arteau[5]),
- **Active scan++** (Autor: James Kettle[11]),
- **Server-Side Prototype Pollution Scanner** (Autor: Gareth Heyes[10]),
- **NoSQLi Scanner** (Autor: Gabriele Gristina[9]),
- **Web Cache Deception Scanner** (Autor: Johan Snyman [19]).

Narzędzie Semgrep wyróżnia się brakiem konieczności instalowania dodatkowych rozszerzeń. Uruchomienie programu z odpowiednią konfiguracją, określającą zestawy reguł do użycia, jest wystarczające. Ponadto, SemGrep oferuje funkcjonalność analizy zależności używanych w aplikacjach, znaną jako Supply Chain, będącą częścią rozwiązania Software Composition Analysis (SCA). Należy jednak pamiętać, że korzystanie z tej funkcji wymaga użycia chmurowego środowiska platformy SemGrep i przesłania kodu na serwery narzędzia. Reguły, które zostały uruchomione z narzędziem:

- **javascript, r2c-security-audit, owasp-top-ten, cwe-top-25**, autorstwa organizacji Semgrep [18].
- **gitlab**, autorstwa organizacji GitLab [8].

Poniżej w Tabeli 5 i 6 zostały przedstawione wyniki uruchomienia narzędzi na testową aplikację internetową.

Parametry\Narzędzie	**SemGrep**	**Burp Suite**
Czas trwania skanu	54s	13:15min
Ilość wykorzystanych reguł i żądań	7855	23732
Wykryte alerty bezpieczeństwa	59	15

Tabela 5. Porównianie paramaetrów narzędzi SemGrep i Burp Suite po rozszerzeniu

Wyniki powyższego porównania wskazują, że po wprowadzeniu ulepszeń obie aplikacje są bardziej efektywne w identyfikowaniu większej liczby podatności. Należy jednak zaznaczyć, że zastosowanie dodatkowych rozszerzeń wydłuża czas trwania skanów w obu narzędziach. W porównaniu do badań przedstawionych w Tabelach 2 i 4, obserwujemy różnice w wykrywaniu podatności związanych z Prototype Pollution przez Semgrep oraz tych związanych z Web Cache Deception przez Burp

Vulnerability\Tools	**SemGrep**	**Burp Suite**
Prototype pollution 1	**Wykryte**	Niewykryte
Prototype pollution 2	**Wykryte**	Niewykryte
Prototype pollution 3	**Wykryte**	Niewykryte
NoSql injection 1	**Wykryte**	**Wykryte**
NoSql injection 2	**Wykryte**	Niewykryte
NoSql injection 3	Niewykryte	**Wykryte**
Web cache deception	Niewykryte	**Wykryte**
Rezultat	**5/7**	**3/7**

Tabela 6. Porównanie wyników skuteczności skanów narzędzi SemGrep i Burp Suite po rozszerzeniu

Suite, co sugeruje, że każde z narzędzi może być skuteczne w wykrywaniu określonych zagrożeń bezpieczeństwa.

Ważne jest podkreślenie, że różnica w wynikach skanowania między Semgrep a Burp Suite wynika z odmiennych metod skanowania, które stosują. Burp Suite pracuje w trybie symulacji ataków na aplikację, wysyłając dużą ilość żądań, co wydłuża czas skanowania, ponieważ narzędzie musi czekać na odpowiedź serwera. Takie podejście pozwala na analizę potencjalnych interakcji między aplikacją a atakującym. Semgrep natomiast skupia się na statycznej analizie kodu, identyfikując możliwe zagrożenia bez przeprowadzania symulacji ataków. Różnice w tych podejściach mają wpływ na liczbę wykrytych alertów bezpieczeństwa oraz na czas potrzebny na wykonanie ska

Rozdział 4

Nienudny tytuł dla teorii

Można też pisać wszystko w jednym pliku, tak jak przyzwyczajają do tego gorsze programy, ale wtedy główny plik będzie bardzo duży i trudniejszy w zarządzaniu.

I to by było na tyle. Kolejny rozdział jest testem ciągłości numeracji rysunków, wzorów i innych elementów graficznych. Za nim jest jeszcze rozdział 6 z podsumowaniem, bibliografia, wykazy, spisy i załączniki.

Rozdział 5

Niebanalny tytuł kolejnego rozdziału

Rozdział 6

Podsumowanie

a

Bibliografia

[1] Abdu-Dabasech, F. i Alshammari, E., „AUTOMATED PENETRATION TESTING: AN OVERVIEW", Department of Computer Science Princess Sumaya University for Technology, Amman, Jordan, 2018.

[2] Akhtar, A., *Popularity Ranking of Database Management Systems*, Dept. of Computer Science, Virtual University of Pakistan, 2023.

[3] Altulaihan, E. A., Alismail, A. i Frikha, M., „A Survey on Web Application Penetration Testing", *Electronics*, t. 12, nr. 5, 2023, ISSN: 2079-9292. DOI: 10.3390/electronics12051229. adr.: https://www.mdpi.com/2079-9292/12/5/1229.

[4] Arteau, O., *Prototype pollution attack in NodeJS application*, Online, Dostęp uzyskano 2024-01-07, 2018. adr.: https://repository.root-me.org/Exploitation%20-%20Web/EN%20-%20JavaScript%20Prototype%20Pollution%20Attack%20in%20NodeJS%20-%20Olivier%20Arteau%20-%202018.pdf.

[5] Arteau, P., *Retire.js*, https://github.com/portswigger/retire-js, [dostęp uzyskano 2024-01-09], 2021.

[6] Eassa, A. M., Elhoseny, M., El-Bakry, H. M. i in., „NoSQL Injection Attack Detection in Web Applications Using RESTful Service", *Programming and Computer Software*, t. 44, s. 435–444, 2018.

[7] Gil, O., *Web Cache Deception Attack*, [dostęp uzyskano 2024-01-07], 2017. adr.: https://www.blackhat.com/docs/us-17/wednesday/us-17-Gil-Web-Cache-Deception-Attack-wp.pdf.

[8] GitLab, *SemGrep Rules*, https://gitlab.com/gitlab-org/security-products/sast-rules/-/blob/main/README.md, [dostęp uzyskano 2024-01-23].

[9] Gristina, G., *NoSQLi Scanner*, https://github.com/portswigger/nosqli-scanner, [dostęp uzyskano 2024-01-09], 2021.

[10] Heyes, G., *Server-Side Prototype Pollution Scanner*, https://github.com/portswigger/server-side-prototype-pollution, [dostęp uzyskano 2024-01-09], 2023.

[11] Kettle, J., *Active Scan++*, https://github.com/portswigger/active-scan-plus-plus, [dostęp uzyskano 2024-01-09], 2023.

[12] Kim, Hee Yeon i in., „DAPP: automatic detection and analysis of prototype pollution vulnerability in Node.js modules", *International Journal of Information Security*, t. 21, nr. 1, s. 1–23, lut. 2022. DOI: 10.1007/s10207-020-00537-0. adr.: https://doi.org/10.1007/s10207-020-00537-0.

[13] Li, S., Kang, M., Hou, J. i Cao, Y., „Tytuł artykułu", w *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, sierp. 2021, s. 268–279. DOI: 10.1145/3468264.3468542.

[14] Mirheidari, S. A., Golinelli, M., Onarlioglu, K., Kirda, E. i Crispo, B., „Web Cache Deception Escalates!", w *31st USENIX Security Symposium (USENIX Security 22)*, sierp. 2022, s. 179–196. adr.: https://www.usenix.org/conference/usenixsecurity22/presentation/mirheidari.

[15] *OWASP in SDLC*, https://owasp.org/www-project-integration-standards/writeups/owaspinsdlc/, [dostęp uzyskano 2024-01-08].

[16] Oyoteyan, D., Milosheska, B., Grini, M. i Cruzes, D., „Myths and Facts About Static Application Security Testing Tools: An Action Research at Telenor Digital", w *Proceedings of the 19th International Conference on Agile Processes in Software Engineering (XP 2018)*, Porto, Portugal: Springer, maj 2018, s. 86–103.

[17] Rangnau, T., Buijtenen, R. v., Fransen, F. i Turkmen, F., „Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines", w *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, 2020, s. 145–154. DOI: 10.1109/EDOC49727.2020.00026.

[18] Semgrep, *Registry*, https://semgrep.dev/explore, [dostęp uzyskano 2024-01-23].

[19] Snyman, J., *Web Cache Deception Scanner*, https://github.com/portswigger/web-cache-deception-scanner, [dostęp uzyskano 2024-01-23], 2017.

[20] Sönmez, F. Ö. i Kiliç, B. G., „Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results", *IEEE Access*, t. 9, s. 25 858–25 884, 2021. DOI: 10.1109/ACCESS.2021.3057044.

[21] ——, „Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results", *IEEE Access*, t. 9, s. 25 858–25 884, 2021. DOI: 10.1109/ACCESS.2021.3057044.

Wykaz skrótów i symboli

Spis rysunków

Spis tabel

Spis załączników

Załącznik 1
Dowód próżni doskonałej

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Załącznik 2
Dowód zera bezwzględnego

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Załącznik 3

Dowód czasu zatrzymanego

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Załącznik 4
Dowód nieskończoności urojonej

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.