Politechnika Gdańska

WYDZIAŁ ELEKTRONIKI
TELEKOMUNIKACJI I INFORMATYKI

Katedra:	Katedra Inżynierii Oprogramowania
Imię i nazwisko dyplomanta:	Alan Turower
Nr albumu:	102185
Forma i poziom studiów:	dzienne magisterskie
Kierunek studiów:	Informatyka

Praca dyplomowa magisterska

Temat pracy:	System wykrywania luk bezpieczeństwa aplikacji webowych
Kierujący pracą:	dr inż. Andrzej Wardziński
Konsultant pracy:	dr inż. Andrzej Wardziński

Zakres pracy:

Praca obejmuje analizę metod ataków na aplikacje webowe poprzez protokół HTTP oraz opis dostępnych narzędzi i technik wykrywania luk bezpieczeństwa. Przeanalizowano, zaprojektowano i wytworzono system wykrywający luki bezpieczeństwa w aplikacjach webowych dostępne poprzez protokół HTTP. System został przetestowany na wytworzonych specjalnie do tego celu środowiskach testowych oraz na rzeczywistych aplikacjach webowych. Został również przetestowany i oceniony przez potencjalnych użytkowników.

Gdańsk 2009

OŚWIADCZENIE

Oświadczam, że niniejszą pracę dyplomową wykonałem samodzielnie. Wszystkie informacje umieszczone w pracy uzyskane ze źródeł pisanych oraz informacje ustne pochodzące od innych osób zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami.

Podpis autora

SPIS TREŚCI

SPIS RYSUNKÓW

SPIS TABEL

Wstęp

Cel pracy

W ostatnich latach mamy do czynienia ze znacznym wzrostem znaczenia Internetu w życiu codziennym. Przyczynia się do tego zarówno obniżenie cen za dostęp do globalnej Sieci, jak również rozwój społeczeństwa informacyjnego[1]. Konsekwencją jest stały wzrost ilości usług dostępnych drogą elektroniczną oraz ograniczenie tradycyjnych dróg kontaktów z biznesem na rzecz metod cyfrowych ze względu na ich tańszą obsługę oraz całodobową dostępność.

Rozwój Internetu prowadzi także do rozwoju przestępczości cyfrowej, umożliwionej przez luki organizacyjne i technologiczne. W szczególności powoduje ona znaczne zwiększenie zagrożenia prywatności osób korzystających z usług w Internecie, poprzez narażenie ich prywatnych zasobów elektronicznych na atak. Ataki te mogą przyjmować różną formę, jednak szczególne znaczenie mają ataki na aplikacje webowe. Powszechność tych aplikacji powoduje, że wykorzystywane są na co dzień, często nieświadomie. Obciąża to odpowiedzialnością projektantów, twórców, testerów oraz administratorów aplikacji webowych, gdyż powierzane im są dane osobowe – często milionów ludzi. Każda luka może spowodować ujawnienie ich atakującemu, a każdy udany atak prowadzi do znaczącego spadku zaufania do dostawcy aplikacji. Powoduje to konieczność częstego i skutecznego testowania aplikacji webowych. Dostępne na rynku służące do tego oprogramowanie jest często trudne w obsłudze i niekompletne.

Celem niniejszej pracy jest przeanalizowanie zagrożeń wynikających z istnienia luk w aplikacjach webowych, stworzenie projektu systemu wykrywającego luki dostępne za pomocą protokołu HTTP, implementacja oraz przetestowanie na rzeczywistych aplikacjach webowych. Zagrożenia te stanowią znaczny problem, na który często twórcy oprogramowania zwracają zbyt małą uwagę. Praca ta ma pokazać wielkość zagrożenia i dostarczyć skuteczne, szybkie i łatwe w obsłudze narzędzie pozwalające na wykrycie słabych punktów tworzonej aplikacji webowej jeszcze przed jej wdrożeniem.

Dodatkowym celem jest przetestowanie pod względem podatności na luki aplikacji webowych działających w Sieci Komputerowej Osiedla Studenckiego

[1] Społeczeństwo charakteryzujące się przygotowaniem i zdolnością do użytkowania systemów informatycznych, skomputeryzowane i wykorzystujące usługi telekomunikacji do przesyłania i zdalnego przetwarzania informacji. [23]

Politechniki Gdańskiej, której jednym z administratorów jest autor niniejszej pracy. Wytworzona aplikacja pozwoli na sprawdzenie bezpieczeństwa obecnie funkcjonujących systemów, jak również dostarczy wiedzy na temat tworzenia, konfigurowania i utrzymywania ich w przyszłości.

Zawartość pracy

Praca składa się z czterech rozdziałów opisujących kolejne etapy analizy zagrożeń, projektu, wytworzenia i testowania aplikacji oraz podsumowania.

Rozdział 1 omawia i systematyzuje podstawowe pojęcia użyte w niniejszej pracy. Znajduje się tu ogólny opis zagrożeń wynikających z korzystania z Internetu. Omówiono w nim zarówno zalety tworzenia i korzystania z aplikacji webowych, jak również zagrożenia z tym związane. Następnie szczegółowo opisano i przeanalizowano metody wykrywania i usuwania zagrożeń związanych z bezpieczeństwem w Internecie.

Luki bezpieczeństwa aplikacji webowych szczegółowo omówiono w rozdziale 2. W szczególności skupiono się na możliwościach ataków poprzez protokół HTTP. Następnie opisano istniejące narzędzia pozwalające rozwiązać przedstawione problemy, ze szczególnym naciskiem na systemy darmowe. Zakres działania narzędzi został opisany na podstawie przyjętej przez autora klasyfikacji luk bezpieczeństwa aplikacji webowych.

Szczegółowy opis kolejnych kroków inżynierii oprogramowania pozwalających na wytworzenie systemu badającego luki bezpieczeństwa aplikacji webowych dostępne poprzez protokół HTTP został zawarty w rozdziale 3. Znajduje się w nim specyfikacja wymagań na system oraz opisane zostały przypadki użycia. Następnie przedstawiono analizę i projekt wytwarzanej aplikacji. Składają się na niego szczegółowy schemat i opis metody testowania aplikacji webowych, opis podsystemów wraz z diagramami UML oraz opis schematu używanej bazy danych. Przedstawiono również technologie, w których został wytworzony system oraz środowiska testowe. Opisano także implementację systemu i jego użytkowanie.

W rozdziale 4 zaprezentowano metodę testowania aplikacji oraz szczegółowo opisano wytworzone i wykorzystane środowiska testowe. Zostały przedstawione wyniki zarówno testowania wytworzonego systemu na specjalnie przygotowanych środowiskach testowych, jak również testów systemów użytkowanych w Sieci Komputerowej Osiedla Studenckiego Politechniki Gdańskiej. Przedstawiono też wyniki porównania skuteczności testowania tychże systemów z wybranymi, wcześniej opisanymi, narzędziami.

Wnioski końcowe płynące z wytworzenia aplikacji, główne osiągnięcia oraz możliwości rozwoju zostały opisane w podsumowaniu.

Konwencje notacyjne

W niniejszej pracy przyjęto następujące konwencje notacyjne:

- Kursywą zaznaczono wyrazy napisane w języku obcym, np.: ang. *Denial of Service*,
- Czcionką Arial oznaczono dłuższe cytowane fragmenty tekstu, np. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Hendrerit. Sed scelerisque tellus quis metus. Phasellus ac nibh volutpat eu, elit. Aenean venenatis pede. Fusce urna quis massa.
- Czcionką Courier New oznaczono fragmenty kodu, np.: `new Class();`

1. Bezpieczeństwo w Internecie

1.1. Podstawowe pojęcia

Poniżej zostały przedstawione podstawowe pojęcia używane w pracy.

1.1.1. Aplikacja webowa

Program komputerowy dostępny przez Internet, który ma bezpośredni kontakt z użytkownikiem, realizuje zadania dla użytkownika. [1] Zazwyczaj celem aplikacji webowych jest dostarczanie usług jej użytkownikom w trybie ciągłym. Aplikacja webowa składa się z wielu stron ułatwiając tym samym jej obsługę.

1.1.2. Atak

Działanie mające na celu włamanie się do aplikacji webowej, zmianę sposobu jej działania lub zablokowanie dostępu do niej.

1.1.3. Haker

Osoba, która korzystając z nieudokumentowanych lub nieznanych powszechnie cech programów usiłuje włamać się do sieci lub systemu komputerowego, głównie za pośrednictwem Internetu, w celu udowodnienia swoich umiejętności, zaszkodzenia komuś lub osiągnięcia korzyści materialnych. [1]

1.1.4. Incydent bezpieczeństwa teleinformatycznego

Zdarzenie zagrażające albo naruszające bezpieczeństwo teleinformatyczne, tj. działania obejmujące przede wszystkim naruszenie poufności, integralności, dostępności zasobów sieciowych, w sposób świadomy lub nieświadomy. Do incydentów bezpieczeństwa teleinformatycznego zaliczamy m.in. ataki na aplikacje webowe. [2]

1.1.5. Protokół HTTP

Ang. *Hypertext Transfer Protocol*, protokół komunikacyjny używany przy dostępie do stron WWW, w tym do aplikacji webowych. [1]

1.1.6. Serwer aplikacji WWW

Platforma oprogramowania pozwalająca na udostępnienie w Internecie aplikacji webowych oraz świadczenie innych usług. Umożliwia oddzielenie logiki biznesowej od usług dostarczanych wraz z platformą.

1.1.7. Strona WWW

Dokument hipertekstowy wyświetlany za pomocą przeglądarki WWW, przesyłany przez Internet za pomocą protokołu HTTP. Może zawierać tekst, pliki graficzne, dźwiękowe i inne. [1]

1.1.8. Wirus komputerowy, robak sieciowy

Najczęściej złośliwe oprogramowanie, które w sposób celowy powiela się bez zgody i wiedzy użytkownika. Wirus komputerowy, w przeciwieństwie do robaka sieciowego, wymaga do swojego funkcjonowania i rozprzestrzeniania się tzw. „nosiciela" w postaci poczty elektronicznej lub programu komputerowego. Wirusy i robaki najczęściej, oprócz rozpowszechniania się, umożliwiają atakującemu dostęp do systemu ofiary (tzw. „koń trojański") lub niszczą zasoby w systemie.

1.1.9. Włamanie

Bezprawne uzyskanie dostępu do systemu lub sieci komputerowej mające miejsce po pokonaniu zabezpieczeń lub wykorzystaniu podatności oprogramowania zainstalowanego na komputerze lub podatności sieci.

1.1.10. World Wide Web (WWW)

System powiązanych ze sobą informacji udostępnianych w sieci Internet w postaci stron WWW, przesyłanych za pomocą protokołu HTTP. Do wyszukiwania i przeglądania tych informacji służą programy zwane przeglądarkami WWW. [1]

1.2. Istniejące problemy związane z bezpieczeństwem w Internecie

Od powstania Internetu znacznie wzrosła jego popularność i liczba użytkowników. W 2006 roku dostęp do Internetu posiadało 36% polskich gospodarstw, zaś 22% gospodarstw było wyposażonych w Internet szerokopasmowy. [3]

W przypadku firm wartości te są znacznie wyższe: wg danych ze stycznia 2008 roku 93% przedsiębiorstw w Unii Europejskiej, zatrudniających co najmniej dziesięciu pracowników lub więcej, miało dostęp do Internetu, zaś 81% przedsiębiorstw UE[2] miało dostęp do Internetu szerokopasmowego. W Polsce odsetek ten jest znacznie mniejszy, jednak także wysoki – dostęp do Internetu szerokopasmowego miało 59% przedsiębiorstw. [4]

[2] Unia Europejska.

Obecnie znaczna liczba firm udostępnia swoją ofertę w Internecie, ale też coraz więcej z nich zaczyna działać tylko w sferze Internetu, oferując tam swoje usługi. W styczniu 2008 wśród przedsiębiorstw z UE posiadających stronę w Internecie, 57% udostępniało katalogi swoich produktów lub listę cenową, 26% umożliwiało zamawianie produktów online, 26% przedsiębiorstw umieszczało na stronie ogłoszenia o pracę lub przeprowadzało procesy rekrutacyjne online, a 10% zapewniało możliwość płatności internetowej. [4]

1.2.1. Zagrożenia związane z korzystaniem z Internetu

Gwałtowny rozwój usług internetowych spowodował także gwałtowny rozwój przestępczości internetowej, którą można zaobserwować na wielu poziomach. Unia Europejska wyróżnia następujące przestępstwa komputerowe, z których zdecydowana większość możliwa jest dzięki Internetowi: [5]

- oszustwo związane z wykorzystaniem komputera,
- fałszerstwo komputerowe,
- zniszczenie danych lub programów komputerowych,
- sabotaż komputerowy,
- „wejście" do systemu komputerowego przez osobę nieuprawnioną,
- „podsłuch" komputerowy,
- bezprawne kopiowanie, rozpowszechnianie lub publikowanie programów komputerowych prawnie chronionych,
- bezprawne kopiowanie topografii półprzewodników,
- modyfikacja danych lub programów komputerowych,
- szpiegostwo komputerowe,
- używanie komputera bez zezwolenia,
- używanie prawnie chronionego programu komputerowego bez upoważnienia.

The European Computer Security Incident Response Team Network (eCSIRT.net) definiuje następujące kategorie incydentów związanych z bezpieczeństwem:

Tabela 1. Incydenty bezpieczeństwa teleinformatycznego wg typów (źródło: [6])

Typ incydentu	Opis
Oszustwa komputerowe	Zdarzenia polegające na celowym czerpaniu korzyści poprzez nieuprawnione wykorzystanie zasobów sieciowych (informacji, systemu), np. kradzież tożsamości (podszycie się, w tym *phishing*[3]), piractwo, plagiat, użycie e-maila w nielegalnych łańcuszkach finansowych itp.
Atak na bezpieczeństwo informacji	Zdarzenia polegające na naruszeniu poufności i integralności informacji, które najczęściej stały się zagrożone poprzez wcześniejsze przejęcie systemu lub przechwycone podczas transmisji danych (np. podsłuch/przechwycenie informacji, zniszczenie lub modyfikacja informacji)
Atak na dostępność zasobów	Zdarzenia polegające na blokowaniu dostępności zasobów sieciowych (systemu/sieci, informacji) poprzez wysyłanie dużej ilości danych, które mogą doprowadzić do odmowy świadczenia usług; destrukcja, zakłócenie. (np. ataki typu DoS/DDoS, w tym mailbombing)
Włamania sieciowe	Zdarzenia polegające na uzyskaniu nieautoryzowanego dostępu do systemu (sieci), tj. wtargnięcie, naruszenie systemu zwykle poprzez wykorzystanie znanych podatności (luk) systemu itp.
Próby włamań	Zdarzenia polegające na podejmowaniu działań w celu uzyskania nieautoryzowanego dostępu do systemu (np. wielokrotne próby nieuprawnionego logowania, „łamanie haseł", próby naruszenia systemu lub zakłócania usług przez wykorzystywanie podatności)
Gromadzenie informacji	Zdarzenia polegające na podejmowaniu działań w celu uzyskania informacji o systemie (sieci) zmierzające do nieautoryzowanego dostępu (np. skanowanie portów, inżynieria społeczna, podsłuch)
Złośliwe oprogramowanie	Zdarzenia polegające na rozpowszechnianiu szkodliwych (złośliwych) programów takich jak np. wirus, robak, koń trojański, dialer, program szpiegujący, zwykle powodujących przeciążenia, destrukcję i destabilizację systemu (sieci)

[3] Rodzaj ataku opartego na inżynierii społecznej polegający na próbie wyłudzenia poufnych danych poprzez podszycie się pod zaufaną osobę lub instytucję.

Obraźliwe i nielegalne treści	Zdarzenia polegające na rozpowszechnianiu obraźliwych i nielegalnych treści (np. pornografia dziecięca, przemoc, spam, treści obraźliwe, groźby i inne naruszenia zasad i reguł w sieci Internet) oraz innych niebezpiecznych (szkodliwych) treści
Inne	Zdarzenia, które nie mieszczą się w wymienionych kategoriach ze względu na typ incydentu itp.

Według CERT Polska, w 2008 roku w Polsce miały miejsce następujące incydenty związane z bezpieczeństwem w Internecie:

Tabela 2. Statystyki zgłoszonych incydentów w Polsce w 2008 roku [7]

Typ/Podtyp incydentu	Liczba	Suma-typ	Procent-typ
Oszustwa komputerowe	10	731	40,70
Nieuprawnione wykorzystanie zasobów	5		
Naruszenie praw autorskich	316		
Kradzież tożsamości, podszycie się (w tym *phishing*)	400		
Atak na bezpieczeństwo informacji	1	7	0,39
Nieuprawniony dostęp do informacji	5		
Nieuprawniona zmiana informacji	1		
Atak na dostępność zasobów	0	26	1,45
Atak blokujący serwis (DoS)	4		
Rozproszony atak blokujący serwis (DDoS)	22		
Sabotaż komputerowy	0		
Włamania sieciowe	2	81	4,51
Włamanie na konto uprzywilejowane	6		
Włamanie na konto zwykłe	16		
Włamanie do aplikacji	57		
Próby włamań	11	97	5,40
Wykorzystanie znanych luk systemowych	24		
Próby nieuprawnionego logowania	62		
Wykorzystanie nieznanych luk systemowych	0		
Gromadzenie informacji	0	86	4,79
Skanowanie	84		
Podsłuch	0		
Inżynieria społeczna	2		
Złośliwe oprogramowanie	143	276	15,37
Wirus	3		
Robak sieciowy	24		
Kon trojański	104		
Oprogramowanie szpiegowskie	2		
Dialer	0		
Obraźliwe i nielegalne treści	0	482	26,84
Spam	466		
Dyskredytacja, obrażanie	8		
Pornografia dziecięca, przemoc	8		
Inne	10	10	0,56
SUMA	1796	1796	100

Zagrożenia te wynikają nie tylko z nieświadomości i niefrasobliwości użytkowników, których to cech nie da się wyeliminować całkowicie, lecz także z błędnie zaprojektowanego, wytworzonego bądź niewłaściwie przetestowanego oprogramowania.

W związku z rozwojem informatyki i przechowywaniem coraz większej liczby informacji w bazach danych podłączonych do Internetu, problem staje się coraz poważniejszy. Zagrożone są dane osobowe, informacje finansowe, jak również wiele informacji osobistych. Z badań wynika, że jest to powodem zwiększania przez firmy wydatków na bezpieczeństwo danych oraz bezpieczeństwo wykorzystywanych aplikacji. Pomimo trwającego kryzysu organizacje wydadzą ze swoich budżetów w tym roku jeszcze więcej środków na bezpieczeństwo, aniżeli w 2008 roku – wynika z raportów Forrester Research. Wydatki na bezpieczeństwo w większych przedsiębiorstwach wzrosną z 11,7 do 12,6 procent. Podobnie sytuacja wygląda w przypadku mniejszych firm. [8]

W większości przypadków użytkownik nie jest w stanie ocenić zabezpieczeń aplikacji, którą wykorzystuje, zatem zapisanie cennych informacji w źle zabezpieczonym miejscu odbywa się niejako nieświadomie. Nakłada to dużą odpowiedzialność na twórców oprogramowania oraz jego administratorów, a także administratorów serwerów, na których działa aplikacja. Nie wszyscy są świadomi tej odpowiedzialności – tak wynika z raportu firmy Napera. Prawie połowa badanych przyznała, że firma nie ma jasnej polityki dotyczącej bezpieczeństwa, zaś ponad 50 procent nie sprawdza, czy firmowe komputery, które mają kontakt z Internetem, są na bieżąco łatane poprawkami. [8]

1.2.2. Zalety aplikacji internetowych

W ostatnich latach coraz popularniejsze i coraz bardziej dostępne są technologie, powalające na tworzenie zaawansowanych aplikacji internetowych, np. AJAX[4] i Flash[5], oraz coraz doskonalsze środowiska wytwórcze takie jak Microsoft Visual Studio, Eclipse IDE, Netbeans IDE. Powoduje to stopniową rezygnację użytkowników z „grubych klientów" na rzecz aplikacji działającej w oparciu o architekturę klient-serwer: działającej na firmowym serwerze (serwerach) aplikacji dostępnej przez przeglądarkę.

[4] Technologia tworzenia aplikacji internetowych, w której interakcja użytkownika z serwerem odbywa się w sposób asynchroniczny, bez przeładowywania całego dokumentu.

[5] Technologia tworzenia animacji z wykorzystaniem grafiki wektorowej rozwijana przez firmę Adobe.

Rozwiązania takie mają szereg zalet, z których najważniejsze to:

- łatwość utrzymania aplikacji – w razie wykrycia błędu konieczne jest naprawienie go jedynie na serwerze, nie zaś na każdym stanowisku roboczym wykorzystującym aplikację;
- łatwość dodania kolejnych stanowisk roboczych – w zasadzie każdy komputer podłączony do Internetu z zainstalowaną przeglądarką internetową może stać się kolejną stacją roboczą wykorzystującą możliwości aplikacji internetowej. Ponadto stacja robocza może znajdować się w dowolnym miejscu na świecie, o ile ma dostęp do Internetu;
- niskie koszty stworzenia stacji roboczej – wymagania techniczne na komputer korzystający z aplikacji internetowej są zazwyczaj bardzo niskie, zaś koszty podłączenia komputera do Internetu ciągle maleją. Siedziba firmy, w której znajdują się stacje robocze, jest zazwyczaj już wyposażona w niezbędną infrastrukturę sieciową i sieć wewnętrzną;
- znane metodologie i technologie tworzenia aplikacji internetowych oraz popularność narzędzi – przyspieszają wytworzenie oprogramowania.

Wszystkie te zalety powodują, że firmy na całym świecie zaczynają doceniać zalety aplikacji webowych i zaczynają je wykorzystywać do następujących celów:

- zarządzania przedsiębiorstwem – systemy kontroli czasu pracy, rozliczania się z wykonanej pracy, motywacji pracowników. Systemy te zazwyczaj działają w sieci wewnętrznej (ang. *Intranet*) przedsiębiorstwa i rzadko dostępne są z sieci publicznej;
- prezentacji przedsiębiorstwa w Internecie oraz sprzedaży usług bądź produktów online – usługi finansowe, sklepy internetowe, witryny przedsiębiorstw;
- zyskowi dzięki reklamom na stronach przyciągających możliwie dużo odwiedzających – portale i wortale[6] internetowe oraz witryny nurtu Web 2.0[7].

Coraz częściej też firmy, które nie są obecne w Internecie (nie reklamują się w nim), postrzegane są jako mniej nowoczesne i innowacyjne.

[6] Portal internetowy dedykowany konkretnej tematyce.

[7] Zbiór technik i wzorców określających sposoby tworzenia witryn internetowych charakteryzujących się wysoką kooperacją z użytkownikiem, wspomagających wymianę informacji i dostępność, w tym otwarte licencje, budujące wokół siebie społeczności użytkowników oraz wytworzonych za pomocą nowoczesnych technologii internetowych [20].

1.2.3. Ataki związane z aplikacjami internetowymi

W związku ze stałym rozwojem technologii aplikacji webowych, pojawiają się również nowe możliwości ataków. Można podzielić je na następujące kategorie:

1.2.3.1. Ataki wykorzystujące błędy w przeglądarce internetowej lub jej niewłaściwą konfigurację

Przeglądarka internetowa jest jedynym oprogramowaniem wymaganym przez aplikację internetową do jej działania na stacji roboczej. Do poprawnego działania oprogramowania jego twórcy często wymagają włączenia obsługi pewnych opcji, jak np. obsługi ciasteczek (ang. *cookies*), języka JavaScript lub określonych technologii, np. Flash. Zdarza się także – szczególnie w przypadku aplikacji wykorzystywanych wewnątrz przedsiębiorstw, gdzie można dostarczyć i wymusić używanie konkretnych i odpowiednio skonfigurowanych przeglądarek – że oprogramowanie jest stworzone i zoptymalizowane do działania na wyłącznie jednej lub jedynie kilku przeglądarkach, zaś inne współpracują niepoprawnie lub w ogóle nie mogą być użyte. Często takie działania przedsiębiorstw mają na celu ograniczenie użycia przeglądarek z aktywnymi opcjami zagrażającymi bezpieczeństwu.

Wybór wspieranej przeglądarki rzadko związany jest z jej bezpieczeństwem (mierzonym jako liczba znalezionych błędów, procent błędów, który został naprawiony lub średni czas naprawienia błędu, w szczególności krytycznego), gdyż najczęściej używaną przez przedsiębiorstwa przeglądarką jest Internet Explorer.

Wg firmy Secunia, zajmującej się bezpieczeństwem, w najpopularniejszych przeglądarkach – w najnowszej wersji w dniu opracowania statystyki – znaleziono oraz naprawiono następująca liczbę błędów wykrytych w okresie 2003-2009:

Rysunek 1. Statystyka błędów dla Internet Explorer 7.x (źródło: [9])

Rysunek 2. Statystyka błędów dla Mozilla Firefox 3.x (źródło: [9])

Rysunek 3. Statystyka błędów dla Opera 9.x (źródło: [9])

Ponieważ wybór przeglądarki, jak również dbanie o jej aktualizację i właściwą konfigurację, leży po stronie użytkownika, jest to źródło potencjalnego zagrożenia, na które nie ma wpływu dostawca oprogramowania. Nawet najlepiej napisane oprogramowanie nie jest w stanie zabezpieczyć nieświadomego i ignorującego zasady bezpieczeństwa użytkownika.

1.2.3.2. Ataki na serwer aplikacji WWW

Serwer aplikacji WWW, tworzący środowisko dla aplikacji WWW, jest źródłem wielu potencjalnych luk i zagrożeń. Wynika to z faktu, że każda maszyna (serwer) jest inaczej skonfigurowany oraz ma zainstalowane różnorodne aplikacje, co może tworzyć sytuację nieprzewidzianą przez twórców oprogramowania. Większość wykrytych błędów dotyczy błędów programistów danego serwera WWW i – w zależności od dostawcy oprogramowania – są regularnie łatane.

Luki można podzielić w zależności od mechanizmu luki na:

- błędy w obsłudze i parsowaniu poleceń przesyłanych do serwera;
- pozwalające na dostęp do plików znajdujących się poza środowiskiem działania serwera (np. poprzez zastosowanie w ścieżce dostępu znaków wyjścia z podkatalogu ../ (ang. *Discovery Traversal*));
- pozwalające na atak odmowy obsługi (ang. *Denial of Service*, *DoS*);

Znalezione luki można podzielić w zależności od miejsca występowania luki na:

- błędy w serwerze jako takim;
- błędy w systemie obsługi modułów i dodatków (wtyczek, ang. *plug-in*);
- błędy w modułach i dodatkach.

Błędy te mogą prowadzić zarówno do uzyskania dostępu do serwera (w ekstremalnym przypadku na prawach administratora), do zablokowania dostępu do serwera i jego usług (atak DoS, restart serwera), jak również zmiany zawartości bądź działania aplikacji utrzymywanych na serwerze.

Utrzymanie i aktualizacja serwerów należy od ich administratorów, jednak, ze względu na wielkość (liczbę linii kodu) serwerów, ich samodzielne poprawienie jest prawie niemożliwe, nawet jeżeli mają one otwarte źródła. Konieczne jest śledzenie komunikatów autorów serwera oraz stosowanie zalecanych przez nich aktualizacji oraz poprawek w konfiguracji. W chwili obecnej większość aplikacji WWW działa na serwerze Apache, o otwartym kodzie źródłowym, rozwijanym przez niedochodową Apache Software Foundation. Ranking popularności stosowania wg badań ze stycznia 2009 roku, przeprowadzonych przez Netcraft wygląda następująco:

Tabela 3. Najczęściej używane serwery WWW na świecie (źródło: [10])

Dostawca	**Produkt**	**Szacunkowa liczba utrzymywanych stron w sztukach**	**Szacunkowa liczba utrzymywanych stron w procentach**
Apache	Apache	96,531,033	52.05%
Microsoft	IIS	61,023,474	32.90%
Google	GWS	9,864,303	5.32%
nginx	nginx	3,462,551	1.87%
lighttpd	lighttpd	2,989,416	1.61%
Oversee	Oversee	1,847,039	1.00%
Inni	-	9,756,650	5.26%

Wraz z rozwojem technologii zaczęto – oprócz serwerów WWW – wyróżniać także serwery aplikacji WWW, tworzące, w odróżnieniu od serwerów WWW, pełną dedykowaną platformę programistyczną umożliwiającą oddzielenie logiki biznesowej aplikacji od usług dostarczanych przez platformę (bezpieczeństwo, zarządzanie transakcjami, skalowalność, dostęp do baz danych). Najpopularniejsze z nich to:

Tabela 4. Najpopularniejsze serwery aplikacji WWW

Dostawca	Produkt	Platforma
RedHat	JBoss	JEE
IBM	WebSphere	JEE
BEA Systems	BEA WebLogic	JEE
Apache	Tomcat	JEE
Microsoft	platforma .NET	.NET

Właściwa konfiguracja serwerów została omówiona w rozdziale 2.

1.2.3.3. Ataki na aplikacje WWW

Aplikacje webowe mogą być podatne na ataki wynikające z błędów występujących w mechanizmach języka programowania, którego użyto do stworzenia aplikacji. Konieczne jest stosowanie częstych aktualizacji serwera w celu usunięcia źródła błędu zaraz po naprawieniu go przez twórców języka. Atak mogą także umożliwiać błędy projektu i implementacji. Powodują one powstanie luk bezpieczeństwa, umożliwiających atakującemu łatwy dostęp do aplikacji webowej. Luki te zostały omówione w rozdziale 2.

1.3. Metody zapobiegania i zwalczania zagrożeń związanych z bezpieczeństwem w Internecie

Polityka bezpieczeństwa firmy powinna określać metody zapobiegania zagrożeń związanych z bezpieczeństwem w Internecie, a także sposób reagowania w sytuacji wystąpienia ataku. Do działań tych powinny należeć:

- Monitorowanie aktualizacji oprogramowania zewnętrznego, takiego jak przeglądarki i serwery WWW.
- Szczegółowe opisanie konfiguracji oprogramowania znajdującego się na firmowych stajach roboczych.
- Przeprowadzanie testów bezpieczeństwa, w tym dokonywanie kontrolowanych ataków.
- Szkolenie użytkowników i przypominanie im o najważniejszych zasadach bezpieczeństwa.

1.3.1. Obraźliwe i nielegalne treści

Niechciana korespondencja jest coraz większym problemem współczesnego Internetu. Szacuje się, że około 90-95% wszystkich wysyłanych e-maili na świecie to SPAM [11]. Dzieje się tak mimo stosowania coraz dokładniejszych filtrów na

serwerach pocztowych, uzupełniania programów pocztowych w filtry antyspamowe, stosowania dodatkowych filtrów antyspamowych w programach antywirusowych oraz ignorowania przez znaczną większość użytkowników takiej korespondencji.

Obraźliwie i nielegalne treści są także elementem wielu stron internetowych. W celu ochrony użytkownika przed napotkaniem takich treści, wyposaża się przeglądarki internetowe w odpowiednie filtry i parsery, zaś strony zawierające takie dane trafiają na tzw. „czarne listy". Użytkownik przed wejściem na taką stronę jest uprzedzany o niewłaściwej zawartości.

1.3.2. Złośliwe oprogramowanie

Wirusy i robaki sieciowe rozpowszechniają się za pomocą poczty elektronicznej, sieci wymiany plików oraz niezaufanych stron internetowych. Najpopularniejszym zabezpieczeniem są programy antywirusowe, pozwalające uchronić użytkownika przez pobraniem niebezpiecznego oprogramowania. Dodatkowo wyposaża się przeglądarki internetowe w filtry i parsery, jak również obsługę „czarnych list", dzięki czemu możliwe jest ostrzeżenie użytkownika przed wejściem na stronę mogącą zawierać złośliwe oprogramowanie. Odpowiednie zabezpieczenia posiadają także programy pocztowe, ostrzegające przed otwarciem załączników. Trzeba dodać, że istotna jest edukacja użytkowników, ponieważ filtry te – jeżeli nie są połączone z programem antywirusowym – są mało skuteczne, ze względu na ostrzeganie przed otwarciem każdego pliku (lub każdego pliku o określonych rozszerzeniach), co powoduje zobojętnienie użytkownika na alarmy.

1.3.3. Gromadzenie informacji

Gromadzenie informacji przez osoby do tego nieupoważnione może być przeprowadzone na szereg sposobów:

- podsłuch służy głownie przechwyceniu informacji niezaszyfrowanej, przesyłanej przez Internet. Napastnik może zdobyć nie tylko dane osobiste na temat ofiary lub przechwycić tajemnice firmowe, lecz może także uzyskać dostęp do loginów i haseł – np. podsłuchując transmisję z serwerem pocztowym odbywająca się za pomocą protokołu POP3, którego specyfiką jest przesyłanie każdej informacji otwartym tekstem. Dodatkowo łączenia z takim serwerem mogą odbywać się bardzo często (domyślnym ustawieniem wielu programów pocztowych jest 10 minut odstępu pomiędzy próbami połączenia). Zabezpieczeniem przed podsłuchem jest szyfrowanie wszystkich istotnych informacji lub przesyłanie ich za pomocą szyfrowanego połączenia.

- skanowanie jest popularną metodą zdobycia informacji o systemie ofiary oraz odkrycie w nim potencjalnych luk w zabezpieczeniach, ponieważ często na źle zabezpieczonym komputerze działa wiele usług sieciowych. Dodatkowo usługi te bywają niepoprawnie skonfigurowane i nieaktualizowane, co znacznie ułatwia napastnikowi znalezienie w nich słabego punktu i dokonania skutecznego ataku. Zabezpieczeniem jest stosowanie zapór ogniowych (ang. *firewall*), które ukrywają dla atakującego niepotrzebnie otwarte porty oraz raportują ew. próby skanowania. Użytkownicy powinni również być świadomi usług udostępnianych na zewnątrz środowiska oraz aktualizować je możliwie często.
- inżynierię społeczną (socjotechnikę[8]) atakujący stosuje w celu zdobycia informacji niejawnych od nieświadomego użytkownika, łatwowiernie wyjawiającego dane, które trudno byłoby zdobyć innymi metodami. Inżynieria społeczna umożliwia uzyskanie właściwie wszystkich praw dostępu do systemu, zaś jedyną formą zapobiegania jej jest ciągłe szkolenie użytkowników oraz minimalizowanie liczby osób z prawami administracyjnymi do systemu. Ze względu na trudność zapobiegania takim incydentom mówi się, że najsłabszym punktem każdego systemu jest człowiek.

1.3.4. Włamania i próby włamań

Istnieje wiele metod, dzięki którym można starać się uzyskać dostęp do systemu informatycznego. Często wystarczy metodą brutalnej siły (ang. *brute force*) złamać hasło domyślnego użytkownika o prawach administracyjnych (admin, root). Atakujący może wykorzystać luki istniejące w oprogramowaniu – w szczególności dotyczy to powszechnie używanych aplikacji i usług sieciowych, gdyż informacja o wykrytych w nich lukach jest powszechnie dostępna. Szczególnie niebezpieczna jest sytuacja, gdy informacja o luce jest ujawiona przed wydaniem do niej tzw. „łaty" (ang. *patch*). Metodą zapobiegania jest tworzenie możliwie nietrywialnych haseł dostępu (wymuszenie ustawiania takich haseł przez użytkowników) oraz wymuszanie częstej zmiany haseł, w szczególności na kontach o uprawnieniach administracyjnych, a także stałe aktualizowanie systemu.

Problem luk w aplikacjach webowych został szczegółowo omówiony w rozdziale 2.

[8] Działanie zmierzające do uzyskania pożądanego zachowania jednostek, grup społecznych; także kierunek refleksji nad celowymi działaniami społecznymi, przyjmujący ich skuteczność za główne lub jedno z głównych kryteriów analizy. [1]

1.3.5. Ataki na dostępność zasobów

Atak na dostępność zasobów (ang. *Denial of Service*, *DoS*) polega na równoczesnym wysłaniu tak wielu zapytań do aplikacji, by oprogramowanie lub serwer nie były w stanie ich obsłużyć prawidłowo. Zazwyczaj atak prowadzi do przeciążenia maszyny, wykorzystaniu wszystkich wolnych zasobów lub zapełnienia systemu plików. Działanie takie może mieć na celu zarówno zablokowanie usługi, prowadzące do strat finansowych oraz utraty wiarygodności dostawcy usługi, jak również – w skrajnych przypadkach – przejęcia kontroli nad systemem. Atak DoS można przeprowadzić z pojedynczego komputera lub w sposób rozproszony:

- rozproszony atak na dostępność zasobów (ang. *Distributed Denial of Service*, *DDoS*) polegający na jednoczesnym wysyłaniu żądań obsługi z wielu maszyn, co znacząco zwiększa siłę ataku;
- rozproszony odbity atak na dostępność zasobów (ang. *Dictributed Reflection Denial of Service*, *DRDoS*) polegający na wysyłaniu do losowo wybranych hostów pakietów SYN[9] ze zmodyfikowanym adresem nadawcy tak, by był nim adres ofiary. Hosty otrzymujące tak zmodyfikowane pakiety masowo odpowiadają pakietami SYN/ACK[10] ofierze, co zazwyczaj powoduje przeciążenie systemu. Metoda ta znacznie utrudnia wykrycie źródła ataku.

Atak DoS zazwyczaj stosowany jest w celu szantażowania firm prowadzących serwisy, na które nałożono wymaganie pracy w trybie ciągłej dostępności (jak np. serwisy aukcyjne) – atakujący zazwyczaj żądają korzyści finansowych za zaprzestanie (lub nie rozpoczynanie) ataku. Zapobieganie tego typu atakom jest trudne, w szczególności gdy przeprowadzany jest w sposób rozproszony. Oprócz stosowania wydajnych maszyn i oprogramowania możliwe jest ciągłe filtrowanie pakietów i odrzucanie tych, których duże ilości nadsyłane są z tego samego źródła. Utrudnieniem dla takich działań jest posiadanie odpowiednio wydajnego sprzętu.

1.3.6. Ataki na bezpieczeństwo informacji

Atak na bezpieczeństwo informacji polega na nieuprawnionym dostępie do informacji lub jej modyfikacji. Atak taki może być następstwem większości z poprzednio opisanych ataków – następuje po uzyskaniu dostępu do systemu, przy czym zazwyczaj wystarczy uzyskać dostęp na niskim poziomie, by mieć dostęp do informacji niejawnych. Zapobieganie takiej sytuacji równa się zapobieganiu poprzednio wymienionym atakom.

---

[9] Atak zalewania pakietami synchronizacji SYN (ang. *SYN flood*).

[10] Pakiet potwierdzenia.

Jak wynika z badań KPMG, liczba osób na świecie, która może w 2009 roku stracić cenne informacje, może sięgnąć 190 milionów – w 2008 roku zdarzenie to dotknęło 92 miliony osób. W ciągu trzech miesięcy (do listopada) 2008 roku na całym świecie 47,8 miliona osób utraciło dane. W ośmiu pierwszych miesiącach 2007 roku było ich o 38 procent mniej (34,5 miliona). [8]

Atak może także umożliwiać błędnie napisana aplikacja internetowa, dając dostęp atakującemu do odczytu, bądź także zapisu (modyfikacji), informacji w systemie. Luki umożliwiające takie działania zostały szczegółowo opisane w rozdziale 2.

1.3.7. Oszustwa internetowe

Oszustwa internetowe są najczęściej notowanym incydentem internetowym ze względu na swoją masową skalę – do przeprowadzenia oszustwa w Internecie często nie jest wymagana specjalistyczna wiedza ani narzędzia. Ten typ incydentu możemy podzielić na:

- naruszenie praw autorskich – jest najczęściej spotykanym incydentem internetowym, często popełnianym przez osoby zupełnie nieświadome. Naruszenie praw autorskich zazwyczaj polega na udostępnianiu w Internecie cyfrowych kopii oprogramowania, filmów lub muzyki, do których nie posiada się praw autorskich. W wielu krajach pobranie takich zasobów jest legalne, jednak oprogramowanie służące do wymiany plików w Internecie zazwyczaj nie pozwala na pobieranie bez jednoczesnego udostępnienia pobranych części pliku. Do zwalczania incydentów związanych z prawami autorskimi zostały powołane specjalne instytucje reprezentujące właścicieli praw, które usiłują zlokalizować osoby udostępniające pliki i powiadomić o tym dostawcę internetowego tejże osoby. Wytwórnie fonograficzne i filmowe, a także wytwórcy oprogramowania, starają się stosować coraz bardziej skomplikowane metody zabezpieczeń produktów. Dodatkowo wiele krajów (w tym kraje Unii Europejskiej) pracuje nad regulacjami prawnymi mającymi powstrzymać naruszanie praw autorskich.
- kradzież tożsamości, podszycie się (w tym *phishing*) wykorzystuje inżynierię społeczną do wmówienia użytkownikowi, iż zaprezentowana mu – specjalnie spreparowana – witryna internetowa jest tą, na którą chciał wejść. Często wykorzystuje się spam do przesłania użytkownikom wiadomości e-mail wyglądających jak wiadomości przesyłane z zaufanej instytucji – np. banku. Wiadomości te zawierają zazwyczaj prośbę o zalogowanie się do systemu lub inną związaną z podaniem danych dostępowych do konta. Znajduje się w niej spreparowany link przekierowujący na podrobioną stronę instytucji, gdzie

użytkownik – w swoim mniemaniu logując się – zostawia atakującemu swój login i hasło. Metodą zapobiegania jest głównie ciągłe szkolenie i ostrzeganie użytkowników. Stosuje się także filtry i „czarne listy" w czytnikach poczty i przeglądarkach, ostrzegające, iż witryna, na której znalazł się użytkownik, może być spreparowana. Szczególnie łatwo poznać to po braku odpowiedniego certyfikatu, którym zazwyczaj zabezpieczone są witryny zaufanych instytucji.

2. Luki w aplikacjach webowych i metody ich wykrywania oraz usuwania

2.1. Istniejące luki w aplikacjach webowych

Aplikacje webowe wymagają wielu różnorodnych zabezpieczeń w celu uniknięcia negatywnych skutków ataków za pomocą protokołu HTTP. Oczywistym jest stosowanie oprogramowania antywirusowego i zapór ogniowych (ang. *firewall*), jak również bezpiecznych metod uwierzytelniania i szyfrowania transakcji, a także wykonywanie kopii zapasowych i dobre zabezpieczenie dostępu do baz danych. Często jednak zapomina się o zasadzie Barry'ego Boehma mówiącej, że naprawienie błędu jest na każdym etapie wytwarzania oprogramowania dziesięciokrotnie wyższe, niż na poprzednim. Wynika z tego, że naprawienie błędów popełnionych na etapie projektowania aplikacji webowej może kosztować od stu do nawet tysiąca razy więcej niż wykrycie ich od razu. Dodatkowo późne wykrycie błędu może spowodować udany atak, którego koszty trudno przewidzieć. Dlatego potrzebny jest zarówno zespół projektowy o wysokim poziomie umiejętności oraz zespół programistyczny znający problematykę tworzenia aplikacji webowych. Konieczne jest również przetestowanie aplikacji internetowej pod kątem występowania w niej błędów przed jej wdrożeniem. Przed udostępnieniem aplikacji należy również sprawdzić podstawowe ustawienia konfiguracji serwera, na którym ma działać aplikacja, aby przekonać się, czy nie ułatwiają one dokonania ataku.

Poniżej zostały przedstawione istniejące typy luk w aplikacjach webowych umożliwiające atak poprzez protokół HTTP, związane z nimi zagrożenia oraz metodologie zapobiegania im.

2.1.1. Konfiguracja serwera udostępniająca informacje przez HTTP

2.1.1.1. Komunikaty i banery serwera

W wielu serwerach konfiguracja tworzona przy instalacji powoduje przedstawianie się serwera, często wraz z jego konfiguracją i listą zainstalowanych dodatków, przy każdym skierowanym do niego zapytaniu. Ustawienie takie jest domyślne między innymi w serwerze Apache, obsługującym ponad 50% stron na świecie (patrz: 1.2.3.2). Napastnik znając nazwę i wersję serwera może łatwo (w szczególności w przypadku nieuaktualnionych wersji) odnaleźć program wykorzystujący znane błędy (ang. *exploit*) w danej wersji serwera. Lukę można sklasyfikować następująco:

- Pozostawienie strony instalacyjnej serwera, dostępnej np. po wpisaniu jego IP, zawierającej zazwyczaj informację jaki serwer zainstalowano, jego wersje oraz – często – informację, że ową stronę należy usunąć – jest to możliwe, gdy aplikacje WWW dostępne na serwerze udostępniane są za pomocą nazw (vhost na podstawie nazwy, nie IP, jest często stosowany ze względu na brak dostępnych IP).

  Metoda testowania: po wpisaniu IP serwera lub jego nazwy (np. nazwa_serwera.przykladowa.domena.pl) powinien wyświetlić się brak dostępu (ew. pusta strona), bądź użytkownik powinien zostać przekierowany na jeden z vhostów obsługiwanych przez serwer, nie zaś stronę startową serwera.
- Pozostawienie sygnatur serwera – po wejściu na nieistniejącą stronę (błąd 404) lub wywołaniu innego błędu, serwer może wyświetlić specjalnie zdefiniowaną stronę bądź domyślny komunikat o błędzie. Domyślnie w serwerze Apache pod komunikatem wyświetlana jest wersja serwera wraz ze wszystkimi uaktywnionymi modyfikacjami. Wyłączenie komunikatu możliwe jest w serwerze Apache przez ustawienie w pliku konfiguracyjnym: *ServerSignature Off*.

  Metoda testowania: należy wejść na nieistniejącą w aplikacji stronę lub wywołać inny błąd serwera WWW. Powinno to spowodować wyświetlenie się komunikatu o błędzie (lub specjalnie przygotowanej strony) nie zawierającej informacji o serwerze WWW.
- Pozostawienie tokenów serwera – domyślnym zachowaniem wielu serwerów jest zwrócenie w nagłówku odpowiedzi (ang. *response*) na żądanie (ang. *request*) informacji o wersji serwera. W przypadku serwera Apache możliwa jest zmiana tej sytuacji na zwracanie jedynie nazwy serwera (*Apache*) za pomocą ustawienia w pliku konfiguracyjnym: *ServerTokens Prod*.

  Metoda testowania: należy przeanalizować odpowiedź na dowolne żądanie HTTP. Odpowiedź nie powinna zawierać informacji o serwerze, lub (jak w przypadku serwera Apache) powinna być zwrócona tylko jego nazwa.

2.1.1.2. Listowanie katalogów

Pozostawienie włączonego listowania katalogów umożliwia przejrzenie zawartości każdego z katalogów, w którym nie umieszczono pliku uznawanego przez serwer za indeks. Może to umożliwić w skrajnym przypadku dostęp do wrażliwych danych umieszczonych w pliku w takim katalogu, zaś w każdym ujawnić strukturę nazewnictwa plików i katalogów. Taka lista pozwala także wyszukiwarkom na zindeksowanie zawartości katalogu i podkatalogów, co w przypadku wycieku informacji powoduje ich dostępność dla osób postronnych, także po wyłączeniu listowania i usunięciu (zmianie lokalizacji) plików. W przypadku serwera Apache

należy się upewnić, czy dyrektywa *Options* w pliku konfiguracyjnym nie zawiera atrybutu *All* lub *Indexes*.

Metoda testowania: przy wykorzystaniu wyszukiwarki lub metodą prób i błędów należy odszukać katalogi, które mają włączone listowanie. Należy wykluczyć katalogi publiczne zawierające treści przeznaczone do udostępnienia.

2.1.2. Nieoczekiwane dane wejściowe

2.1.2.1. Długość przyjmowanych danych (przepełnienie bufora)

Jednym z bardziej niebezpiecznych ataków jest próba przepełnienia bufora (ang. *buffer overflow*), polegająca na pobraniu do wyznaczonego obszaru pamięci (bufora) większej ilości danych, niż zarezerwował na ten cel programista, przez co informacje są nadpisywane na stosie. Niebezpieczeństwo występuje zawsze, gdy nie jest odpowiednio sprawdzana długość wprowadzanych do aplikacji danych. Wszelkie dane wejściowe należy zatem sprawdzać po stronie serwera tuż po ich przyjęciu w taki sposób, by zbyt długi łańcuch został przyjęty lub odrzucony w sposób kontrolowany, ponadto wszystkie elementy tekstowe formularzy (*input*, *textarea*) powinny mieć określoną w formularzu maksymalną długość wprowadzonych danych. Przepełnienie bufora może prowadzić do niespodziewanego działania aplikacji, w przypadku udanego nadpisania informacji na stosie spreparowanymi danymi atakujący może uzyskać dostęp administracyjny do aplikacji.

Metoda testowania: należy do aplikacji wysłać dane, które spełniają wymagania walidacyjne (np. w przypadku numeru telefonu same cyfry), jednak o długości przekraczającej wielkość bufora, który mógł zostać zarezerwowany na przyjmowane dane. Zwykle testujący (atakujący) nie zna rozmiaru zadeklarowanego bufora.

Różne języki programowania i różne serwery aplikacji mają różną podatność na taki atak, np. język C++ (używany na platformie .NET lub poprzez CGI) jest znacznie bardziej podatny na przepełnienie bufora niż PHP.

2.1.2.2. Ciasteczka

Programiści często przechowują dane – pozwalające utrzymać sesję użytkownika lub pozwalające na dostęp do aplikacji bez logowania – w ciasteczkach (ang. *cookies*). Często dane te nie są sprawdzane, gdyż zakłada się, że zostaną otrzymane w postaci zapisanej przez aplikację na dysku. Stwarza to podatność oprogramowania na atak, np. poprzez wprowadzenie znaków specjalnych lub przepełnienie bufora, w przypadku gdy użytkownik zmieni zawartość ciasteczka. Aplikacja powinna zatem sprawdzać, czy wartości wczytane z ciasteczek odpowiadają tym, które mogły być w nich zapisane. Nie należy także zapisywać w ciasteczkach

danych, które po zmianie (tzw. „zatruciu”, ang. *poisoning*) mogą posłużyć użytkownikowi do uzyskania większych przywilejów w trakcie działania aplikacji lub do innego działania, którego nie powinien móc wykonać (np. zapisania praw dostępu do aplikacji).

Metoda testowania: należy zmienić dane w ciasteczku zapisanym przez aplikację na takie, które są niezgodne z logiką aplikacji, zawierające znaki specjalne lub mogą spowodować przepełnienie bufora.

2.1.2.3. Wstrzykiwanie SQL

Wstrzyknięcie SQL (ang. *SQL Injection*) jest luką w zabezpieczeniach aplikacji polegającą na nieodpowiednim filtrowaniu lub niedostatecznym typowaniu i późniejszym wykonaniu danych przesyłanych w postaci zapytań SQL do bazy danych. Podatne są na niego systemy złożone z warstwy programistycznej (przykładowo skrypt w PHP, ASP, JSP itp.) dynamicznie generującej zapytania do bazy danych. Zabezpieczenie się przed atakiem polega na dokładnym sprawdzeniu otrzymanych łańcuchów przed przekazaniem ich do bazy danych, czy nie zawierają niedozwolonych znaków, a w szczególności znaków, które mogą wpłynąć na generowane zapytanie SQL, a także sprawdzanie typów (np. sprawdzanie, czy otrzymana wartość jest liczbą, jeśli zapytanie powinno przyjąć liczbę). Zamiennie można użyć oferowanych przez język programowania lub bazę danych funkcji, np. *addslashes()* w PHP lub *mysql_real_escape_string()* w MySQL. Bezpieczniejszą techniką jest użycie mechanizmu tzw. "zaślepek", gdzie zmienne nie są używane bezpośrednio do tworzenia zapytania, a odpowiednie dane dołączane są do zapytania w momencie jego wykonania, po dokonaniu odpowiedniego rzutowania i sprawdzenia (technika często używana w przypadku bazy danych Oracle). Dodatkowo należy ustawić uprawnienia bazy danych na minimalne, pozwalające aplikacji na współpracę (np. zabronić aplikacji usuwania tabel, jeśli nie jest to wymagane).

Metoda testowania: należy przesłać do aplikacji w łańcuchu GET lub POST zmodyfikowaną zmienną, która zgodnie z obserwacją aplikacji może być przekazywana do bazy danych. Zmienna powinna zawierać elementy formatujące ciągu SQL, przy czym, w zależności od używanego przez aplikację silnika bazy danych, mogą one być różne, np. dla bazy MySQL można przekazać:

```
x' OR '1'='1
```

co w przypadku zapytania *select* powinno zwrócić zawartość całej tabeli.

2.1.2.4. Ukryte elementy formularzy

Programiści często przechowują dane logiki aplikacji w ukrytych (ang. *hidden*) elementach formularza. Częstym jest założenie, że owe elementy formularza zwrócą niezmienioną wartość sterującą (jedną z wartości dyskretnych założonych w logice aplikacji). Stwarza to podatność aplikacji na atak np. poprzez wprowadzenie znaków specjalnych lub atak przepełnienie bufora. Koniecznym jest zatem kontrolowanie także wartości zwracanych przez pola, które zgodnie z logiką aplikacji powinny zwracać wyłącznie wcześniej ustawioną wartość dyskretną.

Metoda testowania: należy przesłać do aplikacji żądanie GET lub POST zawierające zmienione na niedopuszczone w logice wartości ukrytych pól formularza, w szczególności o innym typie, zawierające znaki specjalne lub mogące spowodować przepełnienie bufora.

2.1.2.5. Dane o założonym zakresie wartości zwracanych

Programiści często zakładają, że elementy formularza zwrócą jedną z założonych wartości dyskretnych (np. pole typu *checkbox* zwróci prawdę (*true*) lub fałsz (*false*) w zależności od tego, czy zostało zaznaczone) i porównują ją wyłącznie z założonymi wartościami. Stwarza to podatność aplikacji na atak np. poprzez wprowadzenie znaków specjalnych lub atak przepełnienie bufora. W celu ochrony aplikacja powinna kontrolować po stronie serwera dane, które zgodnie z jej logiką mogą zwracać tylko z góry określone wartości.

Metoda testowania: należy przesłać do aplikacji żądanie GET lub POST zawierające zmienione na niedopuszczone w logice wartości, np. dla elementów *checkbox* lub *radio* łańcuch znaków lub wartość mogąca przepełnić bufor.

2.1.2.6. Weryfikacja danych wyłącznie po stronie klienta

Weryfikacja danych po stronie klienta (np. za pomocą języka *JavaScript*) przyspiesza walidację formularza, a także istotnie zmniejsza obciążenie serwera, gdyż nie jest konieczne każdorazowe wysłanie formularza na serwer i oczekiwanie na odpowiedź – co, w przypadku rozbudowanych formularzy powodujących popełnianie przez użytkowników wielu błędów, może powodować znaczne obciążenie. Niekiedy przyjmuje się, że tak sprawdzone dane – gdy walidacja po stronie klienta nie zgłasza błędów – są poprawne i nie kontroluje się ich po stronie serwera, co może prowadzić do przesłania dowolnych wartości do aplikacji w przypadku wyłączenia w przeglądarce obsługi języka *JavaScript* lub wysłania komendy GET lub POST z pominięciem logiki po stronie klienta.

Metoda testowania: w przypadku wykrycia walidacji po stronie klienta można wyłączyć w przeglądarce komunikaty JavaScript i spróbować przesłać niepoprawne wartości, zaś w przypadku testowania automatycznego wychwycić poprawny łańcuch komendy GET lub POST i spróbować wysłać go do aplikacji po podstawieniu niepoprawnych wartości, niezgodnych z logiką aplikacji lub mogących spowodować przepełnienie bufora.

2.1.2.7. Zmiana zachowania aplikacji

Oprócz wyżej wymienionych ataków, nieoczekiwane dane wejściowe (niesprawdzone) mogą powodować zmianę działania aplikacji w inny sposób. Brak kontroli danych wejściowych może np. spowodować przekazanie do aplikacji kodu w języku *JavaScript*, który może wywołać niepożądane akcje, jak np. otwarcie okna z reklamą lub przekierowanie na inną stronę. Aby zapobiec takim atakom, dane powinny być sprawdzane nie tylko przed wprowadzeniem ich do aplikacji, ale w szczególnie wrażliwych miejscach powinny być kontrolowane przed wyświetleniem.

Metoda testowania: wysłanie do aplikacji (przez interfejs lub komendą GET lub POST) ciągu znaków, np. kodu w języku *JavaScript*, w parametrze, który zostanie potem wyświetlony przez oprogramowanie. Aplikacja, jeżeli nawet przyjmie kod, powinna go wyłącznie wyświetlić, nie zaś wywołać. Szczególnie podatne na ten rodzaj ataku są miejsca, w których ze względu na logikę aplikacja musi przyjąć znaki specjalne takie jak znaki mniejszości, większości, cudzysłowy i średniki.

2.1.3. Hasła i sesje

2.1.3.1. Brak zabezpieczenia

Naturalnym jest zabezpieczenie dostępu do wrażliwych danych hasłem. Nie zawsze jednak aplikacja poprawnie kontroluje dostęp – może się zdarzyć, że jedynie strona główna lub strona główna i strony wymagają hasła, by mieć dostęp do ich zawartości, zaś poszczególne pliki składowe można wyświetlić bez autoryzacji. W oczywisty sposób sytuacja taka stwarza niebezpieczeństwo ujawnienia ważnych informacji osobom nieupoważnionym.

Metoda testowania: oprócz próby wyszukania katalogów z dostępnym listowaniem zawartości (patrz: 2.1.1.2) można użyć wyszukiwarki w celu sprawdzenia zaindeksowanych treści, do których teoretycznie nie mamy dostępu – np. występujących w katalogu *admin*. W przypadku wyszukiwarki Google wystarczy zapytanie:

```
site:www.aplikacja.pl/admin +filetype:*
```

2.1.3.2. Brak mocnych haseł

Innym błędem może być brak sprawdzania siły haseł dostępu, w szczególności zabezpieczających dostęp do kont administracyjnych lub mających dostęp do wrażliwych danych. Konta takie powinny mieć także ustawioną maksymalną liczbę prób logowań. W przeciwnym wypadku atakujący ma możliwość złamania hasła i uzyskania dostępu do aplikacji.

Metoda testowania: przy użyciu brutalnej siły (ang. *brute force*) można próbować słownikowo złamać hasło dla najczęściej występujących loginów o dostępie administracyjnym (*admin, root*). Test można wykonać także np. przy użyciu aplikacji *John The Ripper*.

2.1.3.3. Przejęcie sesji, przejęcie żądania HTTP

Przejęcie sesji (ang. *session hijacking*) oznacza typ ataku, w którym atakujący usiłuje uzyskać dostęp do istniejącej sesji użytkownika. Pozwala na uzyskanie nieuprawnionego dostępu do systemu z prawami użytkownika, od którego przechwycono tzw. klucz sesji, czyli pseudolosowy ciąg znaków identyfikujący sesję użytkownika. Dane dotyczące sesji są często przechowywane w ciasteczkach, co umożliwia jej przechwycenie za pomocą ataku na zawartość ciasteczek (patrz 2.1.2.2) lub technik XSS (patrz: 2.1.2.7). Rzadziej dane dotyczące sesji przesyłane są za pomocą adresu URL lub atrybutów GET lub POST żądania HTTP. Najbezpieczniejsza metodą ochrony jest szyfrowanie przesyłanych danych pomiędzy użytkownikiem a serwerem, w szczególności klucza sesji.

Przejęcie żądania HTTP (ang. *Cross Site Request Forgery, XSRF*) jest wysłaniem żądania HTTP w imieniu innego, zalogowanego użytkownika. Atak możliwy jest po przejęciu sesji użytkownika lub w przypadku, gdy wykonanie akcji możliwe jest bez weryfikacji tożsamości wykonującego.

Metoda testowania: należy sprawdzić, czy klucz sesji generowany jest za pomocą algorytmu minimalizującego ryzyko odgadnięcia klucza sesji.

2.1.4. Inne luki

2.1.4.1. Lokalizacja aplikacji na serwerze, lokalizacja plików z wrażliwymi danymi

Aplikacje WWW często umieszczane są na serwerze w taki sposób, aby miały dostęp do wszystkich (lub większości) katalogów serwera. Aplikacja powinna być umieszczona w wyizolowanym środowisku (tzw. *chroot*), skąd nie będzie miała dostępu do niewymaganych do jej działania gałęzi systemu plików. Dodatkowo można wyizolować katalogi, do których nie będzie możliwy dostęp z poziomu zapytania HTTP

– zalecane jest trzymanie w nich plików z wrażliwymi danymi, do których użytkownik nie musi mieć dostępu. Brak takiej konfiguracji serwera może umożliwić dostęp napastnika do plików na serwerze. W wypadku poważnych luk może się to wiązać z przejęciem przez atakującego kontroli nad całą maszyną.

Metoda testowania: można próbować uzyskać dostęp do katalogu plików dodając odpowiednią ilość komend przejścia do katalogu wyżej w hierarchii („../") w stosunku do wywoływanego adresu lub, jeśli jest możliwość przekazania lub zatrucia danych, przekazania ich w momencie, gdy aplikacja sięga po plik na dysku.

2.1.4.2. Atak na dostępność zasobów

Aplikacja pozwalająca na zbyt wiele interakcji jednego użytkownika w danym momencie ze sobą podatna jest na atak zablokowania dostępu (ang. *Denial of Service*), patrz: 1.3.5. Szczególnie wrażliwe są miejsca, gdzie możliwość przesłania na serwer danych ma użytkownik niezalogowany (np. formularze kontaktowe), bądź aplikacje, gdzie użytkownik może często wysłać dużą ilość danych na serwer (np. fora internetowe). Rozwiązaniem jest dodanie użytkownikom niezalogowanym (a w skrajnych przypadkach także zalogowanym) tokena CAPTCHA (ang. *Completely Automated Public Turing test to tell Computers and Humans Apart*), czyli obrazka z wyświetlonymi losowo znakami, które musi podać użytkownik, aby zrealizować akcję. Inna metodą, często stosowaną na formach internetowych, jest ustawienie minimalnego czasu pomiędzy akcjami użytkownika (np. pomiędzy wysłaniem kolejnych wiadomości na forum).

Metoda testowania: jeżeli aplikacja pozwala na wysłanie do niej danych przez użytkownika niezalogowanego (dotyczy to w szczególności prostych do wypełnienia formularzy), należy sprawdzić, czy wysłanie danych jest uwarunkowane poprawnym podaniem tokena CAPTCHA. W aplikacjach pozwalających na wysyłanie dużych porcji danych należy sprawdzić, czy możliwe jest wysłanie bardzo dużej ilości danych wiele razy pod rząd w minimalnym odstępie czasu.

2.1.4.3. Komunikaty o błędach

Często aplikacja w razie wystąpienia błędu podaje dane, które programiście pomagają znaleźć go i usunąć, jednak atakującemu ułatwiają włamanie. Warto uzależniać wyświetlanie wszelkich komunikatów diagnostycznych od ustawienia zmiennej mówiącej, czy aplikacja działa w środowisku testowym czy docelowym, lub umieszczać wszystkie komunikaty o błędach w logu aplikacji, użytkownikowi zaś prezentować informację, nie pozwalającą mu na odkrycie luki w aplikacji. Należy także pilnować, by komunikaty występujące po błędnym podaniu danych nie niosły zbyt

wielu informacji, a jednocześnie nie utrudniały poprawnego wypełnienia formularza. Przykładem może być prezentowanie po nieudanej próbie logowania komunikatu informującego, czy podano błędny login czy hasło, gdyż ułatwia to złamanie hasła i dostęp do aplikacji.

Metoda testowania: testów tych w dużej mierze nie da się zautomatyzować, jednak można próbować parsować odpowiedzi serwera po wystąpieniu błędów wyrażeniami regularnymi. Główną metodą testowania jest próba wywołania jak największej liczby błędów lub specyficznych błędów w formularzu i obserwowanie komunikatów o błędach. Można także próbować doprowadzić do awarii aplikacji, podczas której może wyświetlić istotne dane, jak np. ciąg dostępu do bazy danych lub ślad wywołania (ang. *stacktrace*), często zawierający dane, które nie powinny być dostępne dla użytkownika aplikacji.

2.1.4.4. Interfejs CGI

CGI jest znormalizowanym interfejsem umożliwiającym komunikację pomiędzy oprogramowaniem serwera WWW, a innymi programami, które się na nim znajdują. Zazwyczaj program serwera WWW wysyła do przeglądarki statyczne dokumenty HTML. Za pomocą programów CGI można dynamicznie (na żądanie klienta) generować dokumenty HTML uzupełniając je np. treścią pobieraną z bazy danych. Programy CGI są podatne na wszystkie wyżej wymienione błędy, zaś często są pisane w językach[11], których funkcje są mniej odporne na błędy wprowadzanych danych, dlatego należy ich używać ze szczególną ostrożnością. Do wykrywania tych luk można stosować metody opisane we wcześniejszych rozdziałach. Ponadto należy pilnować, by programy CGI, jak cała aplikacja WWW, miały minimalne uprawnienia na serwerze, lub by działały w odizolowanym środowisku (ang. *chroot*).

2.1.4.5. Gotowe aplikacje

W przypadku użycia jako silnika aplikacji gotowego programu, należy – w miarę możliwości – usunąć z niego informacje dotyczące nazwy programu (i jego cech charakterystycznych) lub przynajmniej jego wersji. Dodatkowo należy stosować możliwie szybko łatki (ang. *patche*) udostępnione przez twórców oprogramowania. W przypadku popularnych aplikacji informacja o znalezieniu w niej błędu szybko staje się publicznie dostępna, co ułatwia atakującemu jej wykorzystanie. W szczególności proste jest to w przypadku aplikacji o otwartym kodzie źródłowym ułatwiającym analizę luki i stworzenie metody jej wykorzystania.

[11] Skrypty CGI pisane są zazwyczaj w językach: Perl, PHP, Ruby, Tcl, C, C++, Visual Basic.

Metoda testowania: należy sprawdzić, czy aplikacja podaje (np. w stopce) nazwę wykorzystywanego oprogramowania i jego wersję. Jeżeli występuje wersja można także sprawdzić, czy jest to najnowsza dostępna wersja stabilna.

2.2. Istniejące narzędzia wykrywania luk

Stale powiększająca się liczba aplikacji webowych, ich rosnąca rola, lecz także rosnąca ilość luk bezpieczeństwa powoduje, że przybywa także oprogramowania pozwalającego wykrywać podatności na atak przed znalezieniem ich przez napastnika. Często oprogramowanie takie jest albo wystarczająco uniwersalne, lecz płatne, albo wyspecjalizowane tylko w konkretnych zagadnieniach bezpieczeństwa aplikacji internetowych, więc nie zapewniające pełnego sprawdzenia. Ponadto bywają to programy mało intuicyjne w obsłudze, co zniechęca potencjalnego użytkownika. Poniżej przedstawiono najpopularniejsze oprogramowanie do skanowania aplikacji webowych.

2.2.1. Nessus

Nessus to jeden z potężniejszych skanerów podatności umożliwiający przeprowadzenie ponad 20.000 testów umożliwiających wykrycie luk. Skaner dysponuje własną bazą podatności oraz rozbudowaną bazą wtyczek. Nessus umożliwia aktualizację testów oraz pobranie nowych, jednak dostęp no nich zaraz po ukazaniu się wymaga opłaty. Skaner charakteryzuje się dużą szybkością: przeskanowanie pięciu stacji przy użyciu wszystkich dostępnych wtyczek zabiera około 30 minut. [12]

Tabela 5. Zestawienie możliwości programu Nessus

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	tak
Listowanie katalogów	tak
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	tak
Ciasteczka	tak
Wstrzykiwanie kodu SQL	tak
Ukryte elementy formularza	tak
Dane o założonym zakresie wartości zwracanych	tak
Weryfikacja po stronie klienta	tak
Zmiana zachowania oprogramowania	tak
Hasła	
Zabezpieczenie hasłem	tak
Siła haseł	tak
Inne	
Lokalizacja oprogramowania na serwerze	tak
Ataki DOS	tak
Gotowe oprogramowanie	nie

Aby zrealizować wszystkie wyżej wymienione funkcjonalności, Nessus musi mieć wgrane i uaktywnione odpowiednie wtyczki. Skanowanie nie wykrywa także luk wynikających z logiki przejść między poszczególnymi stronami, lecz jedynie z błędnej konfiguracji serwera lub użycia niewłaściwych bądź podatnych na atak języków i metod – lub ich wersji.

2.2.2. Ettercap

Ettercap to program dla systemów Unix i Linux, który pozwala na podsłuchiwanie (sniffowanie - ang. *skiffing*) sieci lokalnych (LAN). Umożliwia on diagnostykę poprawnego działania sieci oraz na dokonywanie ataków człowieka w środku (ang. *Man In the Middle, MItM*). Program pozwala na wyświetlenie wszystkich aktywnych połączeń oraz podglądanie wysyłanych przez nie komunikatów na wybranym interfejsie sieciowym. Umożliwia także na przechwycenie wszystkich komunikatów, jeżeli nie są one szyfrowane. Kolejną jego możliwością jest wysyłanie

fałszywych pakietów protokołu ARP[12] (ang. *ARP Spoofing*). Ettercap obsługuje także system wtyczek, rozwijający jego możliwości, w szczególności w zakresie modyfikacji i zatruwania pakietów. [12]

Tabela 6. Zestawienie możliwości programu Ettercap

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	tak
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	nie
Wstrzykiwanie kodu SQL	nie
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	nie
Hasła	
Zabezpieczenie hasłem	tak[13]
Siła haseł	tak[14]
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	tak[15]

2.2.3. nmap

nmap (z ang. *network mapper*), program komputerowy autorstwa Fyodora (Gordon Lyon), służący do skanowania portów. Program implementuje wiele różnych technik testowania portów TCP, w tym wspomagające omijanie zapór sieciowych lub platformy Intrusion Detection System[16]. Dodatkowo nmap posiada możliwość

[12] Atak sieciowy pozwalający przechwytywać dane przesyłane w obrębie segmentu sieci lokalnej.

[13] Wymagana jest wtyczka lub analizator podsłuchanych komunikatów.

[14] Ettercap potrafi wychwycić hasło, jeżeli jest przesyłane czystym tekstem (ang. *plaintext*).

[15] Wymagana jest wtyczka.

[16] Oprogramowanie służące do automatycznego wykrywania ataku i reagowania na niego.

identyfikacji systemów operacyjnych na skanowanych hostach. Aplikacja jest skanerem aktywnym, co wiąże się z generowaniem dużej ilości ruchu ułatwiającym skanowanemu wykrycie go. [12]

Tabela 7. Zestawienie możliwości programu nmap

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	tak
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	nie
Wstrzykiwanie kodu SQL	nie
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	nie
Hasła	
Zabezpieczenie hasłem	nie
Siła haseł	nie
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	nie

2.2.4. p0f

p0f – pasywny skaner sieciowy stworzony przez Michała Zalewskiego o rozbudowanych możliwościach, umożliwiających m.in. rozpoznawanie systemu operacyjnego, wykrywanie obecności zapór ogniowych i NAT-u, czy wykrywanie czasu pracy (ang. *uptime*). p0f jest skanerem pasywnym, dzięki czemu generuje niewielką ilość ruchu utrudniając wykrycie skanowania. [12]

Tabela 8. Zestawienie możliwości programu p0f

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	tak
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	nie
Wstrzykiwanie kodu SQL	nie
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	nie
Hasła	
Zabezpieczenie hasłem	nie
Siła haseł	nie
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	nie

2.2.5. Snort

Snort to sieciowy sensor służący do wykrywania i zapobiegania atakom na dany węzeł sieci, dostępny na licencji wolnego oprogramowania. Snort posiada szeroki zakres mechanizmów detekcji ataków oraz umożliwia, w czasie rzeczywistym, dokonywanie analizy ruchu i rejestrowanie pakietów, wykrywanie ataków i anomalii, takich jak przepełnienia bufora, skanowanie portów typu stealth, ataki na usługi WWW, SMB, próby wykrywania systemu operacyjnego i wiele innych. Może także działać jako niezależny sniffer, rejestrator pakietów lub system IDS[17]. [12]

Ponieważ Snort jest systemem wykrywania i zapobiegania włamaniom (IDS) nie posiada narzędzi testujących zabezpieczenia.

[17] IDS (ang. *Intrusion Detection System*), IPS (ang. *Intrusion Prevention System*) to urządzenie sieciowe zwiększające bezpieczeństwo systemu informatycznego poprzez wykrywanie (IDS) lub wykrywanie i blokowanie (IPS) ataków w czasie rzeczywistym.

2.2.6. Hydra

Hydra jest oprogramowaniem rozwijanym przez niemiecką organizację „The Hacker's Choice" (THC). Jest stosowana do ataków słownikowych w celu znalezienia tzw. słabych kombinacji nazwy użytkownika i hasła. Obecnie Hydra jest głównie wykorzystywana do odzyskiwania haseł w programie Teamspeak.

Tabela 9. Zestawienie możliwości programu Hydra

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	nie
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	nie
Wstrzykiwanie kodu SQL	nie
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	nie
Hasła	
Zabezpieczenie hasłem	nie
Siła haseł	tak
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	nie

2.2.7. Nikto 2

Nikto jest skanerem działającym na zasadzie serwera www udostępnionym na licencji GPL. Program jest w stanie przeprowadzić różnorodne testy aplikacji webowych uwzględniając ponad 3500 potencjalnie niebezpiecznych plików oraz skryptów CGI, ponad 900 wersji serwerów webowych oraz ponad 250 problemów związanych z konkretnymi wersjami serwerów. Jest wyposażony w system obsługi wtyczek oraz aktualizacji automatycznych. [12]

Tabela 10. Zestawienie możliwości programu Nikto2

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	tak
Listowanie katalogów	tak
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	tak
Ciasteczka	nie
Wstrzykiwanie kodu SQL	tak
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	tak
Hasła	
Zabezpieczenie hasłem	tak
Siła haseł	tak
Inne	
Lokalizacja oprogramowania na serwerze	tak
Ataki DOS	tak
Gotowe oprogramowanie	nie

2.2.8. Paros proxy

Paros Proxy jest opartym na Javie narzędziem oceny zabezpieczeń w sieci Web. Wspiera edycję i bieżący podgląd wiadomości HTTP/HTTPS z możliwością zmiany ciasteczek i treści. Posiada rejestrator ruchu sieciowego oraz skaner do testowania ataków typu SQL Injection i Cross Site Scripting (XSS). [12]

Tabela 11. Zestawienie możliwości programu Paros proxy

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	nie
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	tak
Wstrzykiwanie kodu SQL	tak
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	tak
Hasła	
Zabezpieczenie hasłem	nie
Siła haseł	nie
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	nie

2.2.9. WebScarab

Oprogramowanie do analizowania aplikacji, które komunikują się za pomocą protokołów HTTP i HTTPS. WebScarab rejestruje przepływ danych (wysyłane dane i odpowiedzi), które obserwuje pozwalając na ich późniejszą selekcję. [12]

Tabela 12. Zestawienie możliwości programu WebScarab

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	nie
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	tak
Wstrzykiwanie kodu SQL	tak
Ukryte elementy formularza	tak
Dane o założonym zakresie wartości zwracanych	tak
Weryfikacja po stronie klienta	tak
Zmiana zachowania oprogramowania	tak
Hasła	
Zabezpieczenie hasłem	nie
Siła haseł	nie
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	nie

2.2.10. WebInspect

Narzędzie firmy SPI Dynamice jest oprogramowaniem komercyjnym pozwalającym zidentyfikować nieznane luki w zabezpieczeniach. Pozwala także sprawdzić, czy serwer jest prawidłowo skonfigurowany i odporny na takie ataki jak Parameter Injection[18], Cross Site Scripting (XSS), Directory Traversal[19] i innych. [12]

2.2.11. Whisker/libwhisker

Libwhisker jest modułem w Perl nastawiony na testowanie zabezpieczeń serwerów HTTP. Whisker jest skanerem używającym libwhisker. Biblioteka nie zawiera logiki ułatwiającej testowanie, dostarcza jednak wielu użytecznych metod. [12]

2.2.12. Burpsuite

Burpsuite jest oprogramowaniem komercyjnym pozwalającym atakującemu łączyć manualne i automatyczne techniki aby zliczać, analizować oraz atakować

[18] Atak polegający na zatruciu parametrów przesyłanych do aplikacji.

[19] Atak polegający na przesłaniu do aplikacji znaków przejścia do innego katalogu.

i wykorzystywać aplikacje sieci Web. Poprzez działające wspólnie narzędzia pozwala efektywnie zbierać informacje. [12]

2.2.13. Wikto

Wikto jest bezpłatnym narzędziem do sprawdzania przepływu na serwerach sieci Web. Funkcjonalnością jest zbliżony do Nikto, lecz posiada kilka własnych funkcjonalnych innowacji. Został napisany w środowisku Microsoft .NET i do ściągnięcia jego kodu binarnego/źródłowego wymagana jest rejestracja. [12]

Tabela 13. Zestawienie możliwości programu Wikto

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	tak
Listowanie katalogów	tak
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	tak
Ciasteczka	nie
Wstrzykiwanie kodu SQL	tak
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	tak
Hasła	
Zabezpieczenie hasłem	tak
Siła haseł	tak
Inne	
Lokalizacja oprogramowania na serwerze	tak
Ataki DOS	tak
Gotowe oprogramowanie	nie

2.2.14. Acunetix Web Vulnerability Scanner

Acunetix Web Vulnerability Scanner jest oprogramowaniem komercyjnym, które automatycznie sprawdza aplikacje Web w celu określenia podatności na ataki typu SQL Injection oraz Cross Site Scripting (XSS). Oprogramowanie posiada komfortowy GUI i możliwość tworzenia profesjonalnych raportów audytowych odnośnie zabezpieczeń stron. [12]

2.2.15. Watchfire AppScan

Watchfire AppScan jest oprogramowaniem komercyjnym pozwalającym na testowanie zabezpieczeń aplikacji w trakcie rozwoju. Program skanuje wiele powszechnych podatności, takich jak Cross Site Scripting (XSS), zmiany parametrów, manipulacja ukrytymi polami, przepełnienie bufora i inne. [12]

2.2.16. ratproxy

Półautomatyczne, głównie pasywne narzędzie do audytu aplikacji webowych stworzone na licencji Apache License 2.0. Ratproxy wykrywa potencjalne problemy bezpieczeństwa i błędy w wykorzystaniu wzorców projektowych związanych z bezpieczeństwem bazując na ruchu wygenerowanym przez użytkownika. Wykrywa problemy związane m.in. włączaniem skryptów, prezentowaniem zawartości, niewystarczającymi zabezpieczeniami XSRF i XSS.

Tabela 14. Zestawienie możliwości programu ratproxy

Konfiguracja serwera udostępniająca informacje przez HTTP	
Komunikaty i banery serwera	nie
Listowanie katalogów	nie
Nieoczekiwane dane wejściowe	
Przepełnienie bufora	nie
Ciasteczka	nie
Wstrzykiwanie kodu SQL	nie
Ukryte elementy formularza	nie
Dane o założonym zakresie wartości zwracanych	nie
Weryfikacja po stronie klienta	nie
Zmiana zachowania oprogramowania	tak
Hasła	
Zabezpieczenie hasłem	nie
Siła haseł	nie
Inne	
Lokalizacja oprogramowania na serwerze	nie
Ataki DOS	nie
Gotowe oprogramowanie	nie

3. Aplikacja HttpValider

3.1. Wymagania

Wytworzone oprogramowanie powinno spełniać jawnie określone wymagania. Każde z niżej przedstawionych wymagań ma określone priorytety określające potrzebę jego realizacji w końcowej wersji aplikacji. Mogą one przyjąć następujące wartości:

- wymagane – aplikacja nie będzie pożyteczna bez zaimplementowania tego wymagania,
- oczekiwane – aplikacja będzie pożyteczna bez zaimplementowania tego wymagania, jednak jest ono oczekiwane,
- opcjonalne – nie musi zostać zaimplementowane.

3.1.1. Wymagania funkcjonalne

Aplikacja powinna spełniać następujące wymagania funkcjonalne:

Tabela 15. Wymaganie funkcjonalne FUN_1

Wybór aplikacji i testów	
ID:	FUN_1
Opis:	Użytkownik powinien móc wybrać aplikację webową do przetestowania (przez podanie jej adresu WWW lub podanie pliku z zapisanymi uprzednio wykonanymi żądaniami GET i POST) oraz testy, które zostaną na niej wykonane. Testy do wyboru mogą różnić się w zależności od wybranej metody testowania.
Priorytet:	wymagane
Powiązania:	-

Tabela 16. Wymaganie funkcjonalne FUN_2

Prezentacja wyników w formie raportu	
ID:	FUN_2
Opis:	Wyniki testowania aplikacji powinny zostać przedstawione w formie czytelnego raportu, określającego poziom zagrożenia wynikający z każdej znalezionej luki.
Priorytet:	wymagane
Powiązania:	-

Tabela 17. Wymaganie funkcjonalne FUN_3

Zapisywanie wyników testów	
ID:	FUN_3
Opis:	Aplikacja powinna umożliwić zapis raportu z testowania.
Priorytet:	oczekiwane
Powiązania:	-

Tabela 18. Wymaganie funkcjonalne FUN_4

Eksport raportu do formatu PDF	
ID:	FUN_4
Opis:	Raport z testowania powinien posiadać opcję eksportu do formatu PDF.
Priorytet:	opcjonalne
Powiązania:	-

Tabela 19. Wymaganie funkcjonalne FUN_5

Porównywanie raportów z testowania	
ID:	FUN_5
Opis:	Aplikacja powinna umożliwić porównanie zapisanych raportów z testowania lub wyników testowania z zapisanym raportem tej samej aplikacji webowej i wyświetlić różnice pomiędzy raportami.
Priorytet:	opcjonalne
Powiązania:	FUN_3

Tabela 20. Wymaganie funkcjonalne FUN_6

Utrzymywanie sesji użytkownika	
ID:	FUN_6
Opis:	Aplikacja powinna umożliwić utrzymywanie sesji użytkownika w przypadku testowania wielu kolejnych stron z otrzymanego pliku.
Priorytet:	wymagane
Powiązania:	-

Tabela 21. Wymaganie funkcjonalne FUN_7

Logowanie wykonanych operacji	
ID:	FUN_7
Opis:	Aplikacja powinna umożliwić zapisanie do pliku logu danych o wykonanych w trakcie testowania operacjach (wywołanych żądań HTTP) i szczegółach błędów.
Priorytet:	oczekiwane
Powiązania:	-

3.1.2. Wymagania na realizowane testy bezpieczeństwa

Aplikacja powinna umożliwić wykonanie następujących testów bezpieczeństwa:

Tabela 22. Wymaganie na realizowany test bezpieczeństwa TST_1

Komunikaty i banery serwera	
ID:	TST_1
Opis:	Aplikacja powinna sprawdzać występowanie komunikatów i banerów testowanego serwera zawierających informację o używanym serwerze albo jego wersji w odpowiedziach na żądania HTTP.
Priorytet:	wymagane
Powiązania:	2.1.1.1

Tabela 23. Wymaganie na realizowany test bezpieczeństwa TST_2

Listowanie katalogów	
ID:	TST_2
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest listowanie katalogów, na których operuje testowane oprogramowanie.
Priorytet:	wymagane
Powiązania:	2.1.1.2

Tabela 24. Wymaganie na realizowany test bezpieczeństwa TST_3

Przepełnienie bufora	
ID:	TST_3
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest przepełnienie bufora w testowanym oprogramowaniu.
Priorytet:	oczekiwane
Powiązania:	2.1.2.1

Tabela 25. Wymaganie na realizowany test bezpieczeństwa TST_4

Ciasteczka	
ID:	TST_4
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest wpłynięcie na funkcjonowanie testowanego oprogramowania poprzez zmianę danych w ciasteczkach.
Priorytet:	wymagane
Powiązania:	2.1.2.2

Tabela 26. Wymaganie na realizowany test bezpieczeństwa TST_5

Wstrzykiwanie kodu SQL	
ID:	TST_5
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest wstrzyknięcie kodu SQL do testowanego oprogramowania.
Priorytet:	wymagane
Powiązania:	2.1.2.3

Tabela 27. Wymaganie na realizowany test bezpieczeństwa TST_6

Ukryte elementy formularza	
ID:	TST_6
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest spowodowanie niezaplanowanej przez twórcę reakcji testowanego oprogramowania poprzez zmianę ukrytych elementów formularza.
Priorytet:	wymagane
Powiązania:	2.1.2.4

Tabela 28. Wymaganie na realizowany test bezpieczeństwa TST_7

Dane o założonym zakresie wartości zwracanych	
ID:	TST_7
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest spowodowanie niezaplanowanej przez twórcę reakcji testowanego oprogramowania poprzez zmianę danych o założonych wartościach zwracanych.
Priorytet:	wymagane
Powiązania:	2.1.2.5

Tabela 29. Wymaganie na realizowany test bezpieczeństwa TST_8

Weryfikacja wyłącznie po stronie klienta	
ID:	TST_8
Opis:	Aplikacja powinna sprawdzać, czy poprawność danych w testowanym oprogramowaniu sprawdzana jest wyłącznie po stronie klienta, czy także po stronie serwera.
Priorytet:	wymagane
Powiązania:	2.1.2.6

Tabela 30. Wymaganie na realizowany test bezpieczeństwa TST_9

Zmiana zachowania oprogramowania	
ID:	TST_9
Opis:	Aplikacja powinna sprawdzić, czy możliwy jest atak Cross Site Scripting (XSS) na testowane oprogramowanie.
Priorytet:	wymagane
Powiązania:	2.1.2.7

Tabela 31. Wymaganie na realizowany test bezpieczeństwa TST_10

Dostępność	
ID:	TST_10
Opis:	Aplikacja powinna sprawdzać, czy wszystkie strony testowanego oprogramowania, do których dostęp powinni mieć wyłącznie autoryzowani użytkownicy, są niedostępne dla pozostałych użytkowników.
Priorytet:	oczekiwane
Powiązania:	2.1.3.1

Tabela 32. Wymaganie na realizowany test bezpieczeństwa TST_11

Lokalizacja oprogramowania na serwerze	
ID:	TST_11
Opis:	Aplikacja powinna sprawdzać, czy z poziomu testowanego oprogramowania możliwy jest dostęp do całego katalogu plików serwera.
Priorytet:	oczekiwane
Powiązania:	2.1.4.1

Tabela 33. Wymaganie na realizowany test bezpieczeństwa TST_12

Ataki DoS	
ID:	TST_12
Opis:	Aplikacja powinna sprawdzać, czy możliwe jest częste przesyłanie do testowanego oprogramowania dużej liczby danych.
Priorytet:	wymagane
Powiązania:	2.1.4.2

Tabela 34. Wymaganie na realizowany test bezpieczeństwa TST_13

Gotowe oprogramowanie	
ID:	TST_13
Opis:	Aplikacja powinna sprawdzać, czy testowane oprogramowanie jest instancją gotowego oprogramowania.
Priorytet:	opcjonalne
Powiązania:	2.1.4.5

3.1.3. Wymagania niefunkcjonalne

Aplikacja powinna spełniać następujące wymagania niefunkcjonalne:

Tabela 35. Wymaganie niefunkcjonalne NON_FUN_1

Interfejs użytkownika	
ID:	NON_FUN_1
Opis:	Aplikacja powinna dysponować graficznym interfejsem użytkownika.
Priorytet:	wymagane
Powiązania:	-

3.1.4. Wymagania technologiczne

Aplikacja powinna spełniać następujące wymagania technologiczne:

Tabela 36. Wymaganie technologiczne TECH_1

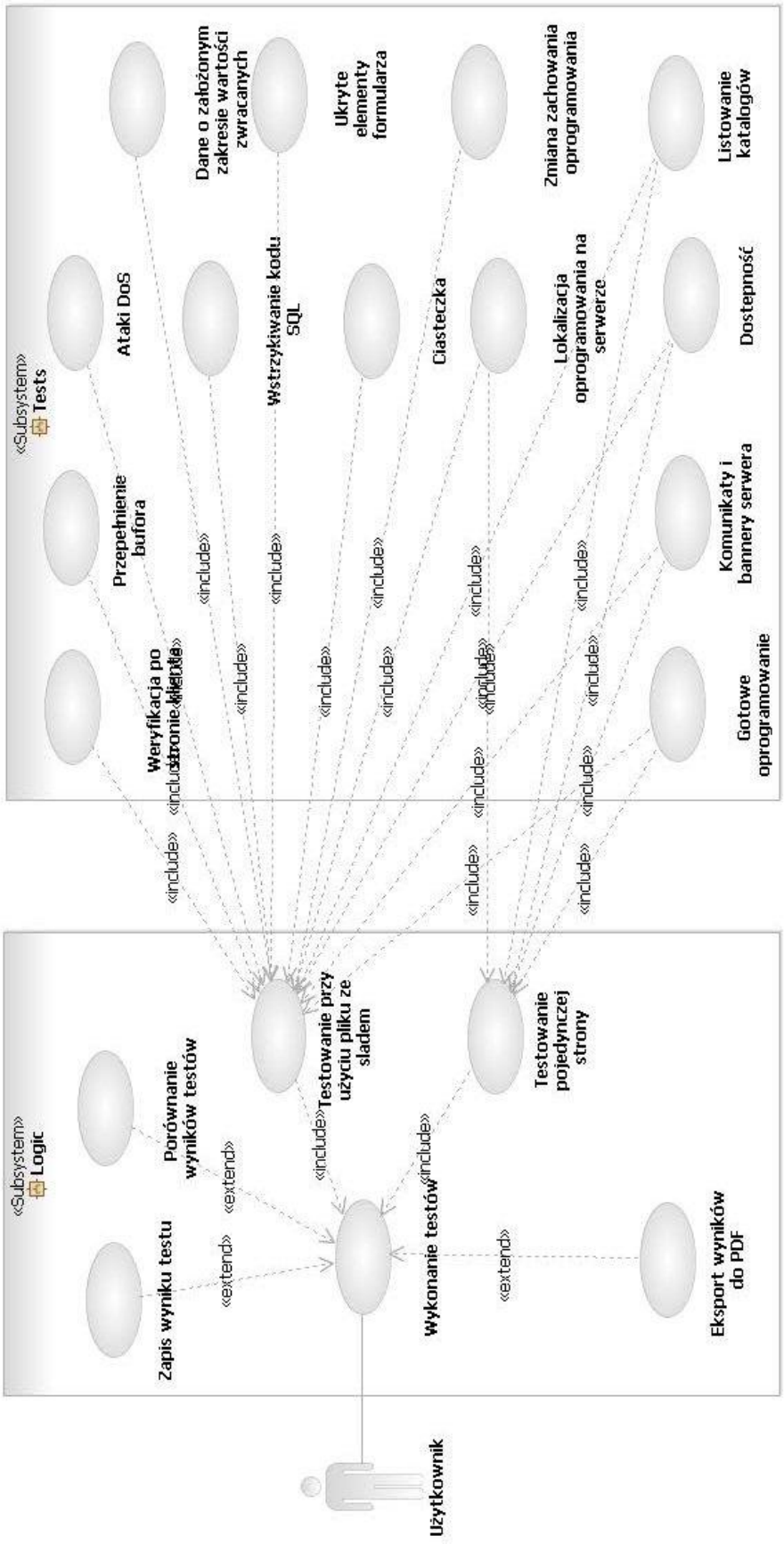
Java	
ID:	TECH_1
Opis:	Aplikacja powinna być stworzona w języku Java, wersja 5 lub wyższa.
Priorytet:	wymagane
Powiązania:	-

Tabela 37. Wymaganie technologiczne TECH_2

Swing	
ID:	TECH_2
Opis:	Graficzny interfejs użytkownika powinien zostać stworzony przy wykorzystaniu biblioteki Java Swing.
Priorytet:	oczekiwane
Powiązania:	NON_FUN_1

3.2. Przypadki użycia

Użytkownik wytworzonej aplikacji powinien mieć możliwość wykonania następujących czynności:



Rysunek 4. Diagram przypadków użycia

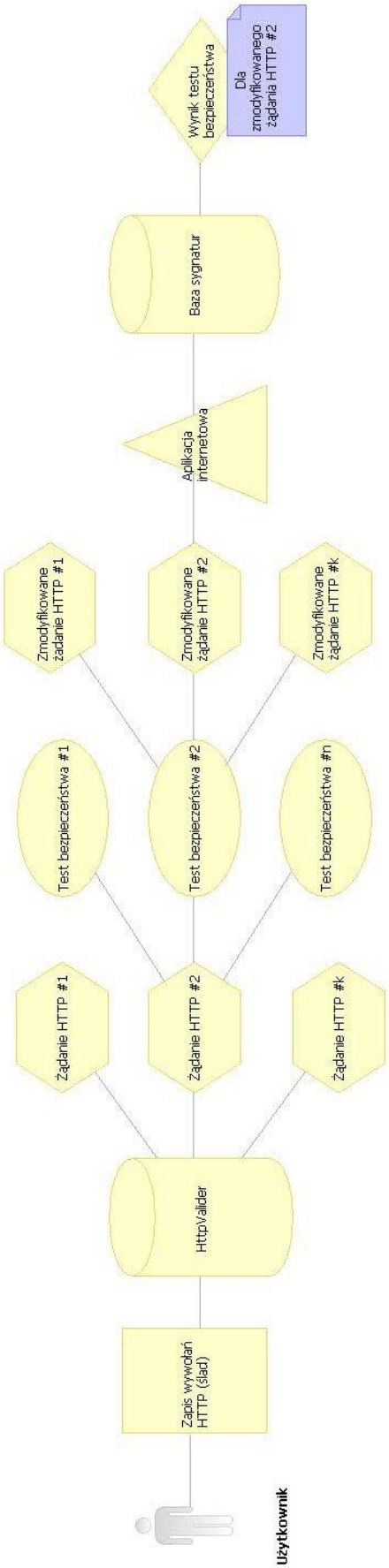
Tabela 38. Opis przypadków użycia

Nazwa przypadku użycia	Realizowana funkcjonalność
Wykonanie testów	Testowanie aplikacji internetowej
Testowanie przy użyciu pliku ze śladem	Testowanie aplikacji wykorzystujące wcześniej zapisany plik z wysłanymi do aplikacji żądaniami HTTP
Testowanie pojedynczej strony	Testowanie aplikacji przez podanie pojedynczego adresu WWW
Komunikaty i banery serwera	Realizacja wymagania TST_1
Przepełnienie bufora	Realizacja wymagania TST_3
Listowanie katalogów	Realizacja wymagania TST_2
Dane o założonym zakresie wartości zwracanych	Realizacja wymagania TST_7
Lokalizacja oprogramowania na serwerze	Realizacja wymagania TST_11
Ukryte elementy formularza	Realizacja wymagania TST_6
Ciasteczka	Realizacja wymagania TST_4
Zmiana zachowania oprogramowania	Realizacja wymagania TST_9
Wstrzykiwanie kodu SQL	Realizacja wymagania TST_5
Ataki DoS	Realizacja wymagania TST_12
Dostępność	Realizacja wymagania TST_10
Weryfikacja po stronie klienta	Realizacja wymagania TST_8
Gotowe oprogramowanie	Realizacja wymagania TST_13
Zapis wyniku testu	Zapisanie wyników testu aplikacji w bazie danych
Porównanie wyników testów	Porównanie uzyskanego wyniku testowania z wynikami testowania tej samej aplikacji zapisanymi w bazie danych
Eksport wyników do PDF	Wyeksportowanie raportu z testowania aplikacji do formatu PDF

3.3. Analiza i projekt

3.3.1. Przebieg testowania aplikacji webowej

Zaprojektowany na potrzeby HttpValidera przebieg testowania aplikacji webowej przedstawia poniższy diagram:



Rysunek 5. Diagram przebiegu testowania

Kolejne kroki przebiegu wyglądają następująco:

3.3.1.1. Zaplanowanie testów i przygotowanie pliku ze śladem

Użytkownik przygotowuje plan testów, określając, które strony aplikacji internetowej zostaną przetestowane. Następnie przechodzi przez wszystkie wymienione strony, starając się wykonać wszystkie dostępne na stronie akcje. Żądania HTTP wygenerowane w trakcie procesu nawigacji po stronie powinny zostać zapisane do pliku śladu – użytkownik może je przechwycić np. za pomocą dodatku Live HTTP Headers do przeglądarki internetowej Firefox (patrz: 3.5.4.1).

3.3.1.2. Uruchomienie aplikacji

Użytkownik podaje lokalizację pliku śladu. W przypadku braku takiego śladu – np. jeżeli użytkownik chce przetestować pojedyncze wywołanie aplikacji internetowej – użytkownik może podać sam adres URL (z parametrami lub bez). Wywołanie podanego adresu zostanie potraktowane jako żądanie GET. Użytkownik zaznacza także testy bezpieczeństwa do przeprowadzenia oraz wybiera ustawienia aplikacji HttpValider.

3.3.1.3. Wywołanie testów

Każde kolejne, jeszcze nie przetestowane, żądanie HTTP przekazane za pomocą pliku śladu lub bezpośrednio powoduje wywołanie kolejnych testów wybranych przez użytkownika.

3.3.1.4. Modyfikacja żądania HTTP

Każdy z testów modyfikuje na jeden lub więcej sposobów otrzymane żądanie i każdą z modyfikacji wysyła do aplikacji internetowej. Modyfikacja może wystąpić pod wpływem zawartości odpowiedzi na poprzednie niezmodyfikowane żądanie, jeśli świadczy ona o możliwości wystąpienia luki na kolejnej stronie.

3.3.1.5. Porównanie odpowiedzi na żądanie HTTP z bazą sygnatur

Każdy z testów posiada własną bazę sygnatur – elementów odpowiedzi HTTP występujących w niej w przypadku wykrycia luki w aplikacji webowej po wysłaniu do niej zmodyfikowanego żądania. Odpowiedź na każde ze zmodyfikowanych żądań porównywana jest z bazą sygnatur. Odpowiedzi – w zależności od testu – porównywane są także z wynikiem na niezmodyfikowane żądanie.

3.3.1.6. Zwrócenie wyniku testowania

Po wysłaniu wszystkich zmodyfikowanych żądań aplikacja zwraca wynik testu bezpieczeństwa dla każdego z pierwotnych żądań.

Wysyłanie zmodyfikowanych żądań dla danego żądania pierwotnego jest przerywane po wykryciu luki bezpieczeństwa. Niektóre testy wykonywane są tylko raz, niezależnie od liczby żądań pierwotnych. Ma to na celu minimalizację liczby wysyłanych żądań HTTP.

3.3.2. Ograniczenia metody

Zastosowana metoda charakteryzuje się kilkoma ograniczeniami:

3.3.2.1. Kompletność testowania

Dla zapewnienia kompletności testowania aplikacji webowej wymagana jest jej bardzo dobra znajomość przez testera. Jest ona konieczna, by zweryfikować wszystkie ścieżki działania aplikacji.

Dodatkowo aplikacja webowa może przyjmować parametry, których nie można przekazać z poziomu działania aplikacji – tzw. tylne drzwi[20] (ang. *backdoor*). Bez wiedzy o takich parametrach nie jest możliwe przetestowanie obsługi przyjmowanych przez nie wartości.

3.3.2.2. Ograniczenia nakładane przez testowaną aplikację

Testowana aplikacja webowa może nakładać ograniczenia na przyjmowane parametry, np. nie pozwalać na wielokrotne utworzenie dokumentu lub konta w systemie. HttpValider w czasie testowania może wielokrotnie wysłać żądanie zawierające te same parametry, jednak zostanie ono obsłużone za drugim i kolejnymi razami inaczej niż pierwsze, co może nie pozwolić na pełne przetestowanie aplikacji.

3.3.2.3. Porównywanie odpowiedzi na żądania

Odpowiedzi na wysłane żądania porównywane są jako pełne otrzymane pliki HTML. Jeżeli strona zawiera treści generowane dynamicznie, np. banery reklamowe, licznik wizyt lub stempel czasowy, kolejne jej odsłony zostaną rozpoznane jako dwie różne strony.

3.3.2.4. Wywoływanie akcji użytkownika

Testowanie może wywołać wiele akcji wymagających reakcji użytkownika, np. podania tokenu CAPTCHA lub aktywacji konta za pomocą linku otrzymanego w adresie e-mail. Wymaga to odpowiedniego zaplanowania testów, współpracy testera z HttpValiderem, a w niektórych wypadkach wyłączenia zabezpieczeń uniemożliwiających automatyczne testowanie aplikacji.

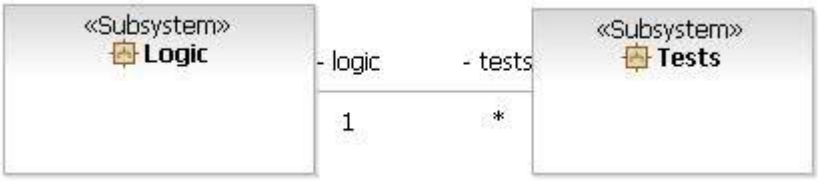
[20] Luka stworzona umyślnie w zabezpieczeniach systemu w celu jej późniejszego wykorzystania, np. nieuprawnionego dostępu do systemu.

3.3.2.5. Dostępność aplikacji

Większość testów dostępnych w HttpValiderze powinna zostać przeprowadzona przed umieszczeniem aplikacji na serwerze produkcyjnym. Jednak różnice pomiędzy serwerami mogą spowodować pojawienie się błędów w aplikacji webowej, np. ze względu na inną wersję parsera używanego języka programowania. Dodatkowo przeprowadzenie niektórych testów, np. dostępności w Google, wymaga, aby aplikacja była dostępna w Internecie przez czas, który pozwoli na jej zindeksowanie.

3.3.3. Podział na podsystemy

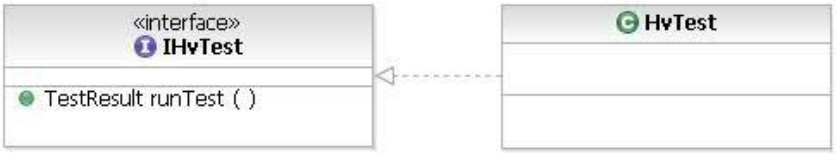
Wytworzona aplikacja składa się z dwóch podsystemów. Ich strukturę przedstawia poniższy diagram:



Rysunek 6. Diagram podsystemów

- Podsystem testów jest implementacją wszystkich wymaganych testów bezpieczeństwa wraz z używanymi przez nie klasami wspomagającymi oraz klasami zawierającymi stałe.
- Podsystem logiki odpowiada za logikę biznesową aplikacji: uruchamianie odpowiednich testów i przekazywanie danych do podsystemu testów, sprawdzanie rezultatów otrzymanych z podsystemu testów i ich przetwarzanie, interfejs graficzny użytkownika, a także operacje dodatkowe takie jak zapisywanie i odczytywanie wyników testów wraz z ich porównywaniem, generowanie raportów i eksport ich do plików PDF.

Na styku pomiędzy podsystemami zaprojektowano interfejs pozwalający podsystemowi logiki na komunikację z podsystemem testów:



Rysunek 7. Interfejs pomiędzy podsystemami

Każdy z testów implementuje interfejs IHvTest, dzięki czemu możliwa jest generalizacja oraz wywołanie testu niezależnie od jego implementacji.

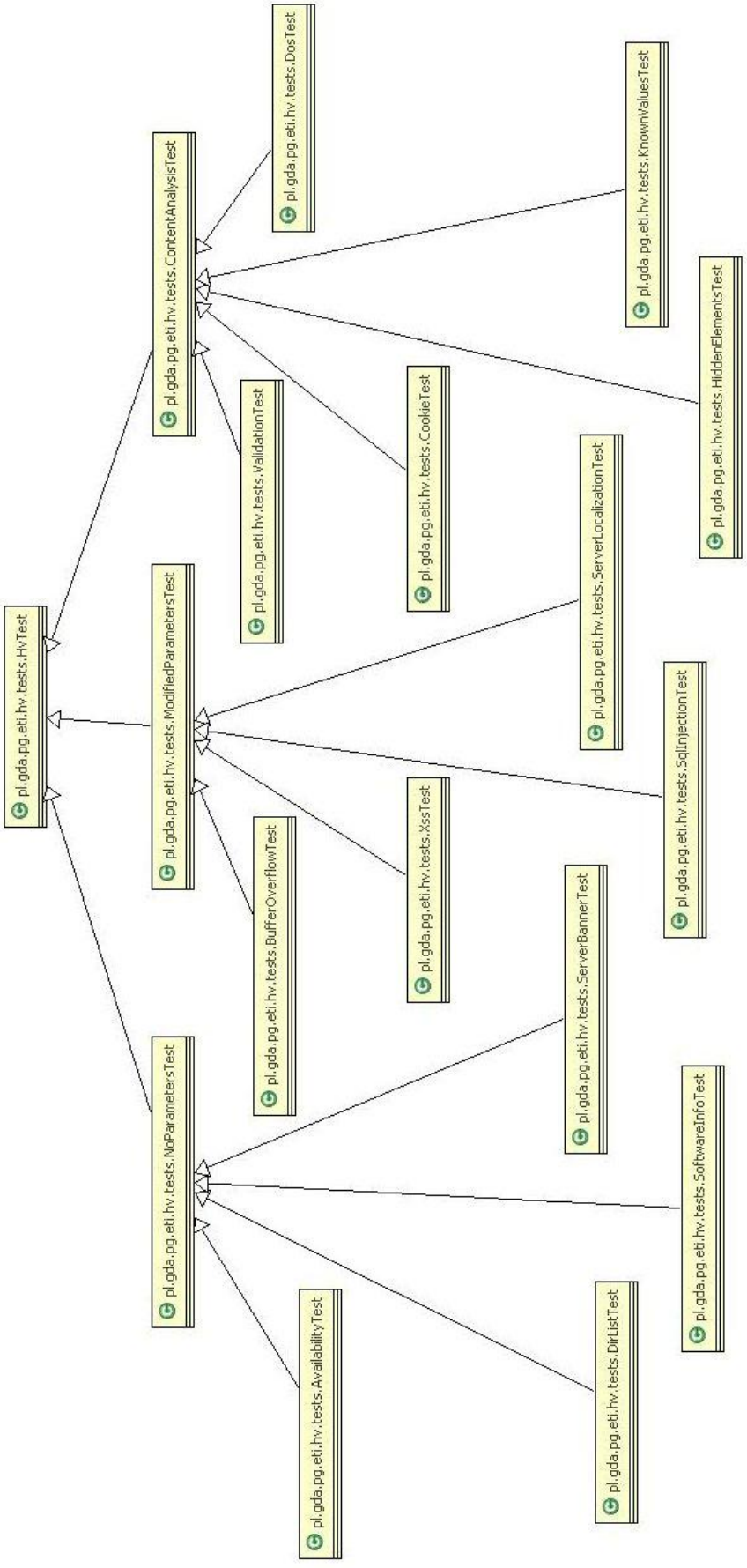
Wytworzenie aplikacji podzielono na dwa przyrosty:

- W trakcie pierwszego zaimplementowany został podsystem testów. Wytworzony prototyp pozwolił stwierdzić, czy wybór języka programowania oraz metodologii jest poprawny.
- W trakcie drugiego przyrostu został zaimplementowany podsystem logiki oraz został on połączony z istniejącym podsystemem testów.

3.3.4. Podsystem testów

Każdy z testów dziedzicząc po klasie HvTest musi oferować konstruktor pozwalający na przekazanie testowi wymaganych do jego pracy danych – w zależności od testu jest to co najmniej adres aplikacji webowej do sprawdzenia, a także dane dotyczące parametrów, ciasteczek, typu żądania HTTP. Każdy z testów implementuje także interfejs IHvTest.

Model klas testów bezpieczeństwa przedstawia się następująco:



Rysunek 8. Model klas testów bezpieczeństwa

3.3.4.1. Klasa HvTest

Jest nadrzędną dla wszystkich testów bezpieczeństwa klasą abstrakcyjną. Oferuje wykorzystywane przez wszystkie testy atrybuty oraz metody, w tym konstruktor, który musi zostać wywołany przez każdy test:

```
public HvTest(String address, List<Parameter> parameters) {
      this.log = new Logger();

      this.testName = testName;

      this.requestData = requestData;

      this.invalidPatterns = new ArrayList<String>();
}
```

Konstruktor inicjuje klasę logującą (*Logger*), zapisuje nazwę wykonywanego testu bezpieczeństwa, przypisuje dane oryginalnego żądania HTTP: adres URL, parametry, ciasteczka, typ żądania oraz inicjalizuje listę nieprawidłowych wzorców poszukiwanych przez test.

3.3.4.2. Klasa NoParametersTest

Podrzędna do klasy HvTest klasa abstrakcyjna będąca nadrzędną dla klas testów niewymagających przesyłania w żądaniu HTTP parametrów. Definiuje wymaganą przez interfejs IHvTest metodę runTest() wraz z metodami pomocniczymi pozwalającymi modyfikować jej działanie testom dziedziczącym bez nadpisywania całej metody. Definiuje także metody używane przez wszystkie klasy dziedziczące.

3.3.4.3. Klasa ModifiedParametersTest

Podrzędna do klasy HvTest klasa abstrakcyjna będąca nadrzędną dla klas testów wysyłających żądania HTTP o zmodyfikowanych parametrach w stosunku do oryginalnego żądania. Definiuje wymaganą przez interfejs IHvTest metodę runTest() oraz metody używane przez wszystkie klasy dziedziczące. Inicjalizuje listę wzorców służących do modyfikacji parametrów oryginalnego żądania HTTP.

3.3.4.4. Klasa ContentAnalysisTest

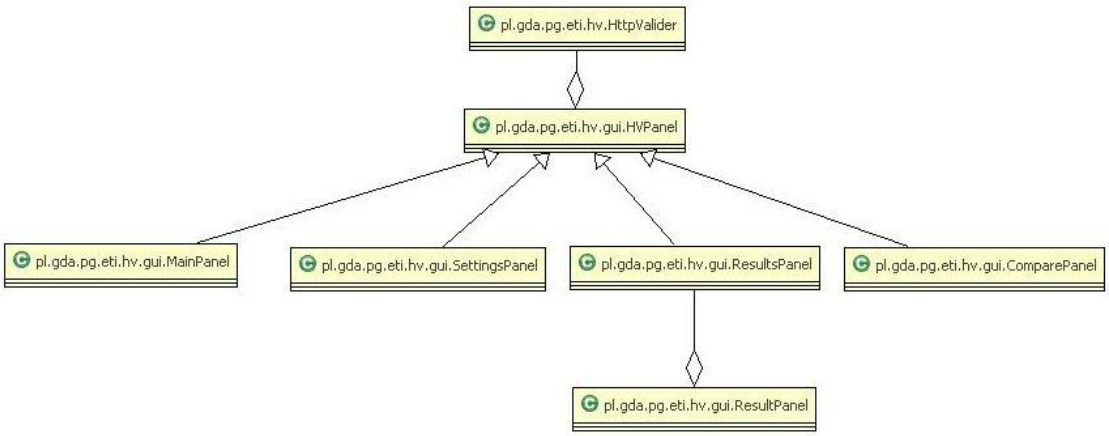
Podrzędna do klasy HvTest klasa abstrakcyjna będąca nadrzędną dla klas testów wysyłających żądania HTTP tylko w wypadku wykrycia w odpowiedzi na oryginalne żądanie elementów wskazujących na podatność na atak. Wysyła odpowiednio zmodyfikowane żądanie i sprawdza, czy odpowiedź świadczy o istnieniu luki. Definiuje wymaganą przez interfejs IHvTest metodę runTest() oraz metody używane przez wszystkie klasy dziedziczące Inicjalizuje listę wzorców służących do modyfikacji parametrów oryginalnego żądania HTTP.

3.3.4.5. Pozostałe klasy podsystemu

Pozostałe klasy przedstawione na modelu są implementacjami poszczególnych testów bezpieczeństwa. Każdy z testów w konstruktorze musi dodać do stworzonych przez klasy nadrzędne struktur wzorce nieprawidłowych zachowań aplikacji lub inne wzorce przewidziane w teście. Dodatkowo każdy z testów może nadpisać metody rodzica, dodając dodatkową funkcjonalność lub zmieniając metodę wysyłania żądań lub porównywania wyników.

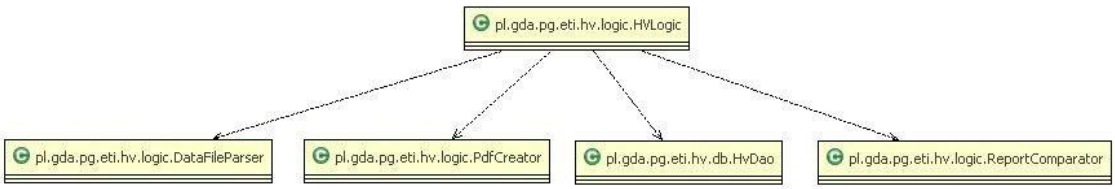
3.3.5. Podsystem logiki

Podsystem logiki składa się z interfejsu użytkownika oraz logiki biznesowej. Model klas interfejsu użytkownika przedstawia się następująco:



Rysunek 9. Model klas graficznego interfejsu użytkownika

Model klas logiki biznesowej przedstawia się następująco:



Rysunek 10. Model klas logiki biznesowej

3.3.5.1. Klasa HttpValider

Jest to główna klasa aplikacji, jako jedyna zawierająca metodę main():

```
public static void main(String args[]) {
	EventQueue.invokeLater(new Runnable() {
		public void run() {
			new HttpValider().setVisible(true);
		}
	});
}
```

Konstruktor klasy tworzy okno aplikacji, wywołuje metodę inicjującą wszystkie

komponenty graficzne (w tym panele przedstawione na diagramie klas) oraz tworzy instancję klasy logiki biznesowej:

```
public HttpValider() {
	super("HttpValider");
	logic = new HVLogic();

	initComponents();
}
```

Ponadto klasa definiuje metody nasłuchujące (ang. *listener*) zdarzeń głównego okna interfejsu użytkownika, w tym wywołania akcji przeprowadzenia testów w logice biznesowej po odpowiedniej akcji użytkownika.

3.3.5.2. Klasa HVPanel

Jest abstrakcyjną klasą panelu użytkownika, po której dziedziczą wszystkie panele okna głównego aplikacji. Konstruktor otrzymuje referencję do głównego okna umożliwiając dwustronną komunikację pomiędzy głównym oknem a jego panelami.

```
public HVPanel(HttpValider hvWindow) {
	this.hvWindow = hvWindow;
}
```

3.3.5.3. Klasa MainPanel

Jest klasą odpowiadającą za główny panel interfejsu użytkownika, definiujący możliwe do przeprowadzenia testy bezpieczeństwa. Jej wywołanie powoduje zainicjowanie elementów graficznych. Implementuje także metody nasłuchujące zdarzeń będących rezultatem akcji użytkownika i odpowiednio modyfikujących panel.

3.3.5.4. Klasa SettingsPanel

Jest klasą odpowiadającą za panel interfejsu użytkownika, pozwalający na modyfikację ustawień aplikacji. Jej wywołanie powoduje zainicjowanie elementów graficznych. Implementuje także metody nasłuchujące zdarzeń będących rezultatem akcji użytkownika i odpowiednio modyfikujących panel.

3.3.5.5. Klasa ResultsPanel

Jest klasą odpowiadającą za panel interfejsu użytkownika, przedstawiający wyniki przeprowadzonych testów. Panel składa się z kolejnych paneli (patrz: 3.3.5.6), z których każdy przedstawia rezultat działania innego testu bezpieczeństwa. Wywołanie klasy powoduje zainicjowanie elementów graficznych. Implementuje także metody nasłuchujące zdarzeń będących rezultatem akcji użytkownika i odpowiednio modyfikujących panel.

3.3.5.6. Klasa ResultPanel

Jest klasą odpowiadającą za panel interfejsu użytkownika, przedstawiający wyniki działania pojedynczego testu bezpieczeństwa. Na panelu może pojawić się wiele kolejnych wyników danego testu, dla kolejnych przekazanych do testu żądań HTTP. Wywołanie klasy powoduje zainicjowanie elementów graficznych. Implementuje także metody umożliwiające dodawanie kolejnych wyników do panelu.

3.3.5.7. Klasa ComparePanel

Jest klasą odpowiadającą za panel interfejsu użytkownika, pokazujący porównanie wyników dwóch różnych sesji testowania przeprowadzonych na tej samej aplikacji webowej – raportów z testowania. Jej wywołanie powoduje zainicjowanie elementów graficznych. Implementuje także metody pomocnicze.

3.3.5.8. Klasa HVLogic

Jest główną klasą logiki biznesowej kierującą przebiegiem zdarzeń w aplikacji. Wywołuje wybrane przez użytkownika i przekazane przez interfejs użytkownika testy bezpieczeństwa. Testy wywoływane są dla kolejnych żądań HTTP, także przekazanych przez użytkownika, zgodnie z wybranymi przez niego ustawieniami.

3.3.5.9. Klasa DataFileParser

Jest klasą odpowiadającą za wczytanie i przetworzenie podanego przez użytkownika pliku zawierającego żądania HTTP wysłane i odebrane od aplikacji webowej podczas używania jej przed testem. Oferuje statyczną metodę odpowiedzialną za przeprowadzenie akcji, zwracającej listę struktur zawierających dane do testów.

3.3.5.10. Klasa PdfCreator

Jest klasą odpowiadającą za przetworzenie raportu wynikowego z testowania aplikacji webowej i wyeksportowanie go do formatu PDF. Klasa oferuje statyczną metodę odpowiedzialną za przeprowadzenie akcji.

3.3.5.11. Klasa HvDao

Jest klasą odpowiadającą za zapisywanie i wczytywanie z bazy danych wygenerowanych przez aplikację raportów z testowania. Klasa oferuje statyczne metody odpowiedzialne za przeprowadzenie akcji.

3.3.5.12. Klasa ReportComparator

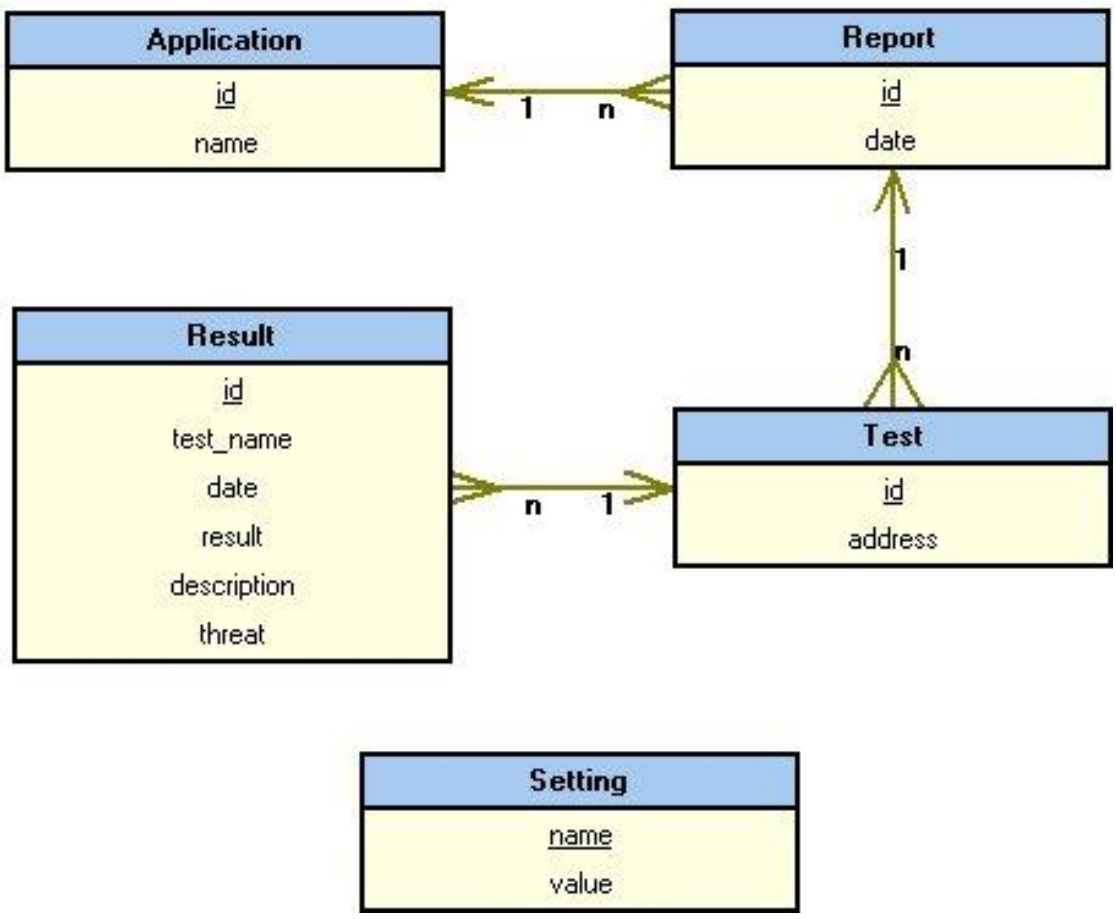
Jest klasą odpowiadającą za przeprowadzenie porównania wygenerowanych przez aplikację raportów z testowania. Klasa oferuje statyczną metodę odpowiedzialną za przeprowadzenie akcji.

3.3.6. Model bazy danych

Aplikacja może na życzenie użytkownika zapisać do bazy danych raport z wykonanego testu, a następnie wczytać go i porównać z innym raportem dotyczącym testowania tej samej aplikacji.

3.3.6.1. Diagram ERD

Diagram ERD bazy danych przedstawia się następująco:



Rysunek 11. Diagram ERD bazy danych

3.3.6.2. Schemat bazy danych

Schemat bazy danych przedstawia się następująco:

- Application (id, name)

  Tablica przechowująca dane dotyczące testowanych aplikacji. Aplikacje rozpoznawane są po adresie URL lub adresie IP.

  - name VARCHAR(64): nazwa aplikacji (jej adres internetowy lub adres IP).

- Report (id, date, ap_id REF Application)
  Tablica przechowująca dane dotyczące pojedynczej sesji testowania aplikacji webowej, która odbyła się konkretnego dnia o konkretnej godzinie.
  - date DATE: data wraz z godziną przeprowadzenia testów.
- Address (id, address, re_id REF Report)
  Tablica przechowująca dane na temat pojedynczego adresu URL wywołanego w ramach sesji testowej.
  - address VARCHAR(64): adres URL testowanej strony.
- Result (id, test_name, date, result, description, threat, ad_id REF Address)
  Tablica przechowująca dane na temat wyniku testowania pojedynczego adresu URL wywołanego w ramach sesji testowej.
  - test_name VARCHAR(64): nazwa testu.
  - date DATE: data wraz z godziną przeprowadzenia testu.
  - result INTEGER: numeryczne oznaczenie wyniku testowania zgodne z oznaczeniami stosowanymi przez aplikację.
  - description VARCHAR(512): opis wyniku.
  - threat INTEGER: poziom zagrożenia wynikający z rezultatu testu.
- Setting (name, value)
  Tablica przechowująca ustawienia HttpValidera.
  - name VARCHAR(32): nazwa ustawienia.
  - value VARCHAR(32): wartość ustawienia.

3.4. Sposób prowadzenia testów

Testy zostały podzielone na trzy grupy (tożsame z trzema klasami dziedziczącymi po klasie HvTest, patrz: 3.3.4). Działanie każdego z nich zostało opisane poprzez zaprojektowany dla niego algorytm. Skuteczność każdego z testów została oceniona jako wysoka, średnia lub niska, w zależności od możliwości wykrycia luki.

3.4.1. Testy nie wymagające przekazania parametrów

Każdy z testów znajdujących się w tej grupie sprawdza ustawienia serwera oraz aplikacji webowej niewymagające przesłania parametrów. Za wyjątkiem testu sprawdzającego, czy użyte są gotowe aplikacje (patrz: 2.1.4.5), testy te przeprowadzane są tylko raz w ciągu całego testowania, ponieważ badane ustawienia powinny być identyczne dla każdej strony i każdego zestawu parametrów.

Tabela 39. Opis algorytmu testu dostępności w Google

Dostępność w Google	
Algorytm testu	1. Sprawdzenie, czy podany adres nie jest adresem lokalnym. Jeżeli tak, test kończy się ostrzeżeniem. 2. Odpytanie wyszukiwarki internetowej Google, czy w zaindeksowanych danych dotyczących aplikacji webowych znajdują się strony zawierające w adresie URL lub tytule którąś z sygnatur świadczących o luce. Przykładowe zapytanie: http://google.interia.pl/szukaj?q=intitle:admin&site:aplikacja.pl Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 3. Sprawdzenie, czy katalogi aplikacji zawierają plik robots.txt, instruujący roboty wyszukiwarek internetowych, które katalogi i pliki można zindeksować. Jeżeli nie, test kończy się ostrzeżeniem. 4. Test kończy się informacją o braku luki.
Ocena skuteczności	Średnia Test może dawać wiele fałszywych alarmów, w szczególności dla aplikacji webowych których treści dotyczą administrowania.
Przykładowa sygnatura świadcząca o luce	admin
Opis luki	2.1.3.1

Tabela 40. Opis algorytmu testu listowania katalogów

Listowanie katalogów	
Algorytm testu	1. Sprawdzenie, czy kolejne katalogi z podanego adresu URL mają włączoną możliwość listowania. Np. dla adresu aplikacja.pl/katalog/strona.php sprawdzane jest listowanie dla adresów aplikacja.pl i aplikacja.pl/katalog. W przypadku wykrycia listowania, test kończy się zgłoszeniem informacji o luce. 2. Jeżeli podany adres nie jest adresem lokalnym, odpytanie wyszukiwarki internetowej Google, czy któraś z zindeksowanych stron aplikacji webowej zawiera sygnaturę świadczącą o luce. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 3. Test kończy się informacją o braku luki.
Ocena skuteczności	Niska Prawdopodobieństwo znalezienia katalogu niezawierającego pliku *index* poprzez przekazane do testowania adresy URL jest niska, tak samo jak zindeksowanie takiego katalogu, niezawierającego danych przeznaczonych do udostępnienia przez wyszukiwarkę Google.
Przykładowa sygnatura świadcząca o luce	Index Of
Opis luki	2.1.1.2

Tabela 41. Opis algorytmu testu gotowego oprogramowania

Gotowe oprogramowanie	
Algorytm testu	1. Wysłanie niezmodyfikowanego żądania HTTP. 2. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 3. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Pokrycie przez test:	Test obecnie pokrywa dwa popularne fora internetowe (phpBB, SMF) oraz dwa popularne systemy zarządzania treścią (WordPress, Joomla!).
Przykładowa sygnatura świadcząca o luce	Powered by PHPBB
Opis luki	2.1.4.5

Tabela 42. Opis algorytmu testu banerów serwera

Banery serwera	
Algorytm testu	1. Wysłanie niezmodyfikowanego żądania HTTP. 2. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 3. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Pokrycie przez test:	Test obecnie pokrywa trzy serwery (Apache, lighttpd, GlassFish) oraz zawiera bazę sygnatur świadczących o podaniu przez serwer informacji o używanym systemie operacyjnym.
Przykładowa sygnatura świadcząca o luce	Apache/*numer_wersji*
Opis luki	2.1.1.1

3.4.2. Testy modyfikujące parametry

Każdy z testów znajdujący się w tej grupie modyfikuje parametry w przekazanym żądaniu w celu wywołania błędu aplikacji.

Tabela 43. Opis algorytmu testu przepełnienia bufora

Przepełnienie bufora	
Algorytm testu	1. Sprawdzenie, czy przekazane żądanie zawiera parametry. Jeżeli nie, test kończy się informacją o braku parametrów. 2. Wysłanie żądania HTTP z kolejno modyfikowanymi parametrami przekazanego żądania. 3. W przypadku wystąpienia wyjątku w trakcie wysyłania żądania, sprawdzenie, czy wyjątek oznacza potencjalną podatność serwera lub aplikacji na przepełnienie bufora. Jeżeli tak, test kończy się ostrzeżeniem. 4. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 5. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 6. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowy zmodyfikowany parametr	Oryginalna wartość parametru znajdującego się w żądaniu powtórzona 10 000 razy.
Przykładowa sygnatura świadcząca o luce	Exception
Opis luki	2.1.2.1

Tabela 44. Opis algorytmu testu zmiany zachowania aplikacji

Zmiana zachowania aplikacji (XSS)	
Algorytm testu	1. Sprawdzenie, czy przekazane żądanie zawiera parametry. Jeżeli nie, test kończy się informacją o braku parametrów. 2. Wysłanie żądania HTTP z kolejno modyfikowanymi parametrami przekazanego żądania. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowy zmodyfikowany parametr	<script>alert();</script>
Przykładowa sygnatura świadcząca o luce	alert();
Opis luki	2.1.2.7

Tabela 45. Opis algorytmu testu wstrzykiwania SQL

Wstrzykiwanie SQL	
Algorytm testu	1. Sprawdzenie, czy przekazane żądanie zawiera parametry. Jeżeli nie, test kończy się informacją o braku parametrów. 2. Wysłanie żądania HTTP z kolejno modyfikowanymi parametrami przekazanego żądania. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowy zmodyfikowany parametr	te'st
Przykładowa sygnatura świadcząca o luce	Query failed
Opis luki	2.1.2.3

Tabela 46. Opis algorytmu testu lokalizacji na serwerze

Lokalizacja aplikacji na serwerze	
Algorytm testu	1. Sprawdzenie, czy przekazane żądanie zawiera parametry. Jeżeli nie, test kończy się informacją o braku parametrów. 2. Jeżeli przekazany adres URL kończy się nazwą pliku, wysłanie żądania HTTP z prefixem dodanym do nazwy pliku. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Wysłanie żądania HTTP z kolejno modyfikowanymi poprzez dodanie prefixu parametrami przekazanego żądania. 6. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 7. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 8. Test kończy się informacją o braku luki.
Ocena skuteczności	Średnia Ujawnienie się luki może wymagać przekazania specyficznego polecenia do serwera.
Przykładowy zmodyfikowany parametr	../../../../../../../../../../
Przykładowa sygnatura świadcząca o luce	No such file or directory
Opis luki	2.1.4.1

3.4.3. Testy modyfikujące parametry na podstawie zawartości prawidłowej odpowiedzi na żądanie HTTP

Każdy z testów z tej grupy analizuje odpowiedź HTTP uzyskaną z wywołania niezmodyfikowanego poprzedzającego lub obecnego żądania i w przypadku wykrycia podatności na lukę modyfikuje parametry w celu wywołania błędu aplikacji.

Tabela 47. Opis algorytmu testu walidacji wyłącznie po stronie klienta

Walidacja wyłącznie po stronie klienta	
Algorytm testu	1. Sprawdzenie, czy odpowiedź na oryginalne żądanie zawiera elementy świadczące o podatności na lukę. Jeżeli tak, następuje modyfikacja wyłącznie parametrów wpływających na te elementy, jeżeli nie, modyfikowane są wszystkie parametry. 2. Wysłanie żądania HTTP ze zmodyfikowanymi parametrami przekazanego żądania. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowa zawartość świadcząca o podatności	value ==
Przykładowy zmodyfikowany parametr	<te'st>%$^
Przykładowa sygnatura świadcząca o luce	Exception
Opis luki	2.1.2.6

Tabela 48. Opis algorytmu testu zatrucia ciasteczek

Zatrucie ciasteczek	
Algorytm testu	1. Sprawdzenie, czy oryginalne żądanie zawiera ciasteczka. Jeżeli nie, test kończy się informacją o braku luki. 2. Wysłanie żądania HTTP z kolejno zmodyfikowaną zawartością ciasteczek przekazanego żądania. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowy zmodyfikowany parametr	<te'st>%$^
Przykładowa sygnatura świadcząca o luce	Exception
Opis luki	2.1.2.2

Tabela 49. Opis algorytmu testu zatrucia ukrytych elementów

Zatrucie ukrytych elementów	
Algorytm testu	1. Sprawdzenie, czy odpowiedź na poprzedzające żądanie zawiera jawnie podane wartości parametrów. Jeżeli tak, wartości tych parametrów są kolejno modyfikowane. Jeżeli nie, przejście do pkt. 5. 2. Wysłanie żądania HTTP ze zmodyfikowanymi parametrami przekazanego żądania. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 1. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Sprawdzenie, czy odpowiedź na poprzedzające żądanie zawiera ukryte elementy. Jeżeli tak, wartości tych parametrów są kolejno modyfikowane. Jeżeli nie, test kończy się informacją o braku luki. 6. Wysłanie żądania HTTP ze zmodyfikowanymi parametrami przekazanego żądania. 7. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 1. 8. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 9. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowa zawartość świadcząca o podatności	hidden
Przykładowy zmodyfikowany parametr	<te'st>%$^
Przykładowa sygnatura świadcząca o luce	Exception
Opis luki	2.1.2.4

Tabela 50. Opis algorytmu testu zatrucia elementów o znanych wartościach zwracanych

Zatrucie elementów o znanych wartościach zwracanych	
Algorytm testu	1. Sprawdzenie, czy odpowiedź na poprzedzające żądanie zawiera jawnie podane wartości parametrów. Jeżeli tak, wartości tych parametrów są kolejno modyfikowane. Jeżeli nie, przejście do pkt. 5. 2. Wysłanie żądania HTTP ze zmodyfikowanymi parametrami przekazanego żądania. 3. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 1. 4. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 5. Sprawdzenie, czy odpowiedź na poprzedzające żądanie zawiera elementy o oczekiwanych wartościach zwracanych. Jeżeli tak, wartości tych parametrów są kolejno modyfikowane. Jeżeli nie, test kończy się informacją o braku luki. 6. Wysłanie żądania HTTP ze zmodyfikowanymi parametrami przekazanego żądania. 7. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 1. 8. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. 9. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowa zawartość świadcząca o podatności	checked
Przykładowy zmodyfikowany parametr	<te'st>%$^
Przykładowa sygnatura świadcząca o luce	Exception
Opis luki	2.1.2.5

Tabela 51. Opis algorytmu testu podatności na atak na dostępności zasobów

Podatność na atak na dostępność zasobów	
Algorytm testu	1. Sprawdzenie, czy odpowiedź na oryginalne żądanie zawiera elementy świadczące o podatności na lukę. Jeżeli tak, następuje modyfikacja wyłącznie parametrów wpływających na te elementy i przejście do pkt. 3. 2. Sprawdzenie, czy oryginalne żądanie zawiera parametry, których wartość przekracza długość 128 znaków. Jeżeli tak, następuje modyfikacja wyłącznie tych parametrów. Jeżeli nie, test kończy się informacją o braku luki. 3. Wysłanie żądania HTTP ze zmodyfikowanymi parametrami przekazanego żądania. 4. Sprawdzenie, czy otrzymana odpowiedź wskazuje na przekierowanie na zdefiniowaną przez użytkownika stronę błędu. Jeżeli tak, powrót do pkt. 2. 5. Sprawdzenie, czy odpowiedź na żądanie HTTP zawiera sygnaturę świadczącą o luce oraz czy odpowiedź na niezmodyfikowane żądanie jej nie zawiera. Jeżeli tak, test kończy się zgłoszeniem informacji o luce. Jeżeli nie, powrót do pkt. 3, aż do wysłania trzech kolejnych żądań. 6. Test kończy się informacją o braku luki.
Ocena skuteczności	Wysoka
Przykładowa zawartość świadcząca o podatności	textarea
Przykładowa sygnatura świadcząca o luce	Too many
Opis luki	2.1.4.2

3.5. Użyte technologie

Do stworzenia aplikacji użyto następujących technologii oraz narzędzi:

3.5.1. Środowisko programistyczne

Aplikacja powstała w zintegrowanym środowisku programistycznym (ang. *Integrated Development Environment*, *IDE*) Eclipse. Jest to platforma stworzona

przez firmę IBM, udostępniona na zasadach wolnego oprogramowania. Projekt jest w chwili obecnej rozwijany przez Eclipse Foundation. Środowisko przeznaczone jest głównie do programowania w języku Java, jednak rozbudowany system wtyczek, zawierający także wsparcie dla twórców nowych modułów powoduje, że możliwe jest wytwarzanie oprogramowania w wielu językach. Dostępne wtyczki oferują także wiele innych funkcji, jak dostęp do repozytorium kodu lub generowanie diagramów. Środowisko zostało wybrane ze względu na jego dużą popularność i bardzo dobrą znajomość przez autora.

Projekt graficznego interfejsu użytkownika został stworzony w edytorze wizualnym (ang. *Designer*) zintegrowanego środowiska programistycznego NetBeans. Jest to platforma stworzona przez czeskich studentów, do której prawa w 1999 roku wykupiła firma Sun Microsystems. Źródła NetBeans są dostępne na licencji Common Development and Distribution License. Środowisko przeznaczone jest głównie do tworzenia aplikacji w języku Java i promowane jako podstawowe narzędzie do tego celu. Zostało wybrane ze względu na brak dobrego edytora wizualnego dostępnego dla środowiska Eclipse.

3.5.2. Język programowania

Cała aplikacja powstała w języku Java 6. Jest to obiektowy język programowania stworzony przez firmę Sun Microsystems. Programy napisane w Javie kompilowane są do kodu bajtowego i wykonywane przez maszynę wirtualną, która musi być zainstalowana na komputerze. Język cechuje się silnym typowaniem. Jest to obecnie jeden z najpopularniejszych języków programowania, bardzo dobrze znany przez autora. Dużym plusem Javy jest jej przenośność pomiędzy systemami operacyjnymi, zaś maszyna wirtualna Javy jest obecnie zainstalowana na większości komputerów używanych na świecie, co czyni ją językiem uniwersalnym.

Rozważane było również stworzenie całej aplikacji lub tylko jej modułu testów w języku Perl. Jest to interpretowany język programowania przeznaczony głównie do pracy z danymi tekstowymi. Jego głównym atutem jest bardzo rozbudowana obsługa wyrażeń regularnych, co czyni go idealnym językiem do przetwarzania żądań HTTP. Perl to wolne oprogramowanie, dostępne pod licencjami GPL i artystyczną. Perl jest dostępny dla wielu systemów operacyjnych, lecz jego naturalne środowisko to Unix i jego pochodne, co znacznie ograniczyłoby przenośność programu, gdyż obecnie niewiele komputerów z systemem Microsoft Windows ma zainstalowany interpreter Perla. Ponadto w trakcie sprawdzania możliwości wyrażeń regularnych języka Java okazało się, że są one wystarczające.

3.5.3. Biblioteki

Aplikacja korzysta z następujących bibliotek:

3.5.3.1. Swing

Swing to biblioteka graficzna używana w języku programowania Java. Jest ulepszoną wersją biblioteki graficznej AWT, którą rozszerza o wiele nowych funkcjonalności. Jest bardzo popularnym rozwiązaniem przy tworzeniu graficznych interfejsów użytkownika oprogramowania tworzonego w języku Java, jest też dobrze znana przez autora.

3.5.3.2. iText

Biblioteka iText umożliwia „w locie" wytwarzanie i modyfikację dokumentów w formacie PDF. Możliwe jest zarówno wytworzenie dokumentu i umieszczenie go w systemie plików, jak również przesłanie go do przeglądarki internetowej. Biblioteka wymaga użycia języka Java w wersji co najmniej 1.4. Jest dostępna za darmo na licencjach MPL oraz LGPL.

3.5.3.3. Apache Derby (JavaDB)

Biblioteka Apache Derby jest małą bazą danych przeznaczoną dla aplikacji napisanych w języku Java. Dzięki wbudowanemu sterownikowi JDBC[21] bibliotekę Apache Derby można zagnieździć w aplikacji, umożliwiając jej korzystanie z bazy danych. Rozwiązanie to zwiększa przenośność oraz minimalizuje liczbę problemów związanych z konfiguracją i administracją bazą danych. Derby spełnia standardy JDBC i SQL. Dostępna jest bezpłatnie na licencji Apache License 2.0. W języku Java, od wersji 6, biblioteka jest wbudowana w język i dostępna pod nazwą JavaDB.

3.5.4. Środowisko testowe

Do zbudowania środowiska służącego do testowania aplikacji w trakcie jej tworzenia użyto następujących języków i narzędzi, poza wyżej wymienionymi.

3.5.4.1. Live HTTP Headers

Live HTTP Headers jest bezpłatnym dodatkiem do przeglądarki internetowej Firefox. Umożliwia podgląd w czasie rzeczywistym nagłówków HTTP przesyłanych

[21] Ang. *Java DataBase Connectivity* – interfejs opracowany przez firmę Sun Microsystems umożliwiający aplikacjom napisanym w języku Java, niezależnie od ich platformy, porozumieć się z bazami danych w języku SQL.

z i do przeglądarki. Rezultat zapisany do pliku używany jest jako plik śladu. Live HTTP Headers należy do projektu mozdev.org.

3.5.4.2. Język PHP

PHP jest obiektowym (od wersji 5), skryptowym językiem programowania dostępnym bezpłatnie na licencji PHP License. Jego rozwojem zajmuje się PHP Group. Najczęściej wykorzystywany jest do przetwarzania skryptów po stronie serwera. Charakteryzuje się modułową budową, dającą możliwości łatwej rozbudowy języka. Często stosowany razem z serwerem WWW Apache oraz bazą danych MySQL – połączenie to znane jest jako platforma AMP. Jego popularność powoduje, że jest idealnym językiem do stworzenia jednej z platform testowych. Ponadto jest bardzo dobrze znany przez autora.

3.5.4.3. Serwer WWW Apache

Apache to otwarty serwer HTTP dostępny dla wielu systemów operacyjnych (m.in. UNIX, GNU/Linux, BSD, Microsoft Windows). Apache jest najszerzej stosowanym serwerem HTTP w Internecie. W połączeniu z interpreterem języka skryptowego PHP i bazą danych MySQL, Apache stanowi jedno z najczęściej spotykanych środowisk w firmach oferujących miejsce na serwerach sieciowych. Jego popularność czyni go idealnym serwerem testowym. Serwer jest dobrze znany przez autora.

3.5.4.4. Serwer WWW lightpd

lighhtpd jest szybkim i bezpiecznym serwerem HTTP zyskującym w ostatnim czasie dużą popularność [13] ze względu na optymalizację dla aplikacji, gdzie krytyczną jest obsługa w czasie możliwie zbliżonym do rzeczywistego. Serwer jest znany przez autora.

3.5.4.5. Serwer aplikacji Glassfish v2

Glassfish jest wieloplatformowym, otwartym serwerem aplikacji webowych dla platformy Java EE. Tworzony jest przez firmę Sun Microsystems i rozpowszechniany na licencji CDDL oraz częściowo GPL. Serwer jest znany przez autora.

3.5.4.6. Baza danych MySQL

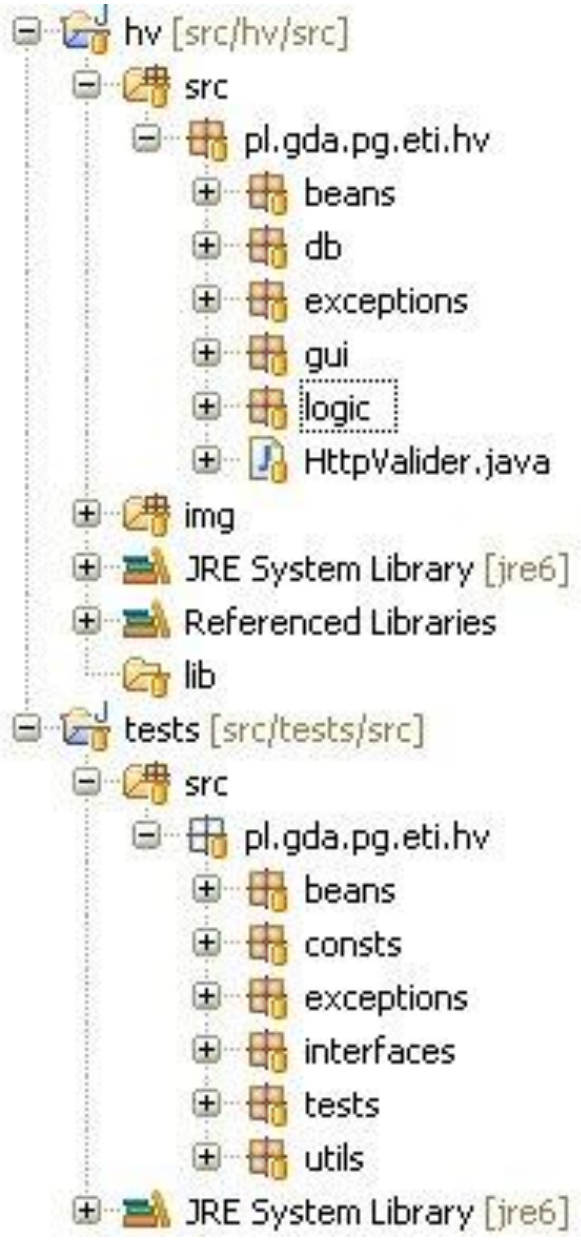
MySQL to wolnodostępny system zarządzania relacyjnymi bazami danych. W chwili obecnej oferuje on rozbudowaną funkcjonalność, zaś jego popularność czyni go idealną bazą testową. MySQL jest dobrze znany przez autora.

3.5.4.7. Baza danych PostgreSQL

PostreSQL jest jednym z najpopularniejszych systemów zarządzania relacyjnymi bazami danych. Dostępny jest na licencji BSD. Oferuje bardzo rozbudowaną funkcjonalność, otrzymał też wiele branżowych wyróżnień. PostreSQL jest znany przez autora.

3.6. Implementacja

Struktura stworzonego kodu wprost odzwierciedla podział na podsystemy przedstawiony w podrozdziale 3.2. Każdy z podsystemów jest oddzielnym projektem Eclipse. Struktura katalogów zachowuje podział na role poszczególnych zbiorów klas:



Rysunek 12. Struktura katalogów aplikacji

- Katalogi beans zawierają klasy przechowujące struktury danych używane w aplikacji zgodnie z konwencją JavaBeans[22].
- Katalogi exceptions zawierają wyjątki (klasy dziedziczące po klasie Exception) stworzone na potrzeby aplikacji.
- Katalog gui podsystemu logiki zawiera klasy tworzące graficzny interfejs użytkownika (patrz: 3.3.5).
- Katalog db podsystemu logiki zawiera klasę implementującą dostęp do bazy danych (ang. *Data Access Object, DAO*).
- Katalog logic podsystemu logiki zawiera klasy definiujące logikę biznesową aplikacji (patrz: 3.3.5).
- Katalog consts podsystemu testów zawiera abstrakcyjne klasy definiujące stałe wartości używane w aplikacji.
- Katalog interfaces podsystemu testów zawiera interfejsy programistyczne umożliwiające połączenie obu podsystemów.
- Katalog tests podsystemu testów zawiera implementację testów bezpieczeństwa (patrz: 3.3.4).
- Katalog utils podsystemu testów zawiera implementację elementów wykorzystywanych przez testy do swojego działania.

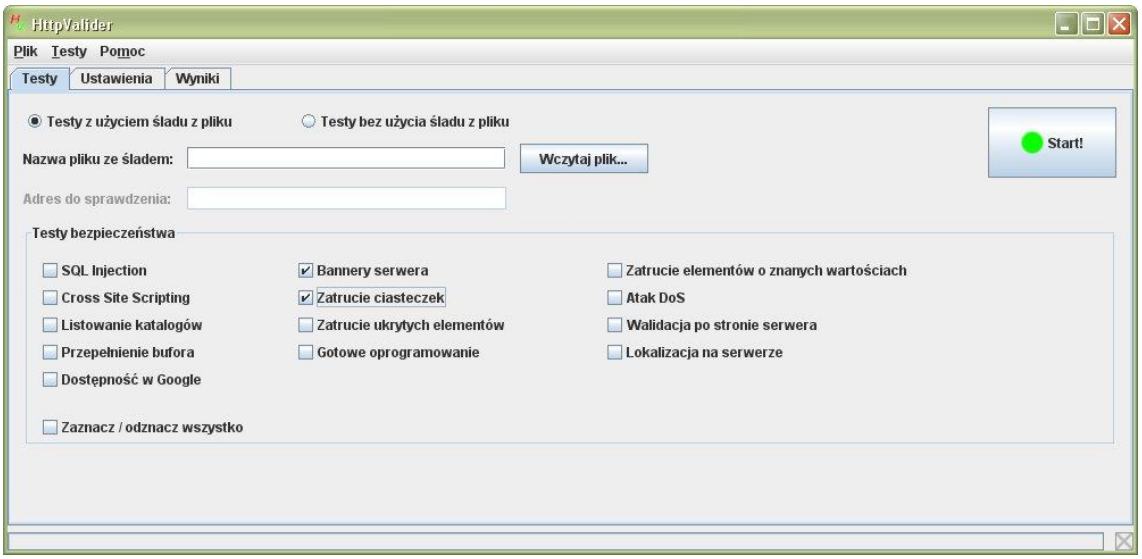
Wszystkie klasy i metody aplikacji zostały skomentowane w konwencji *Javadoc*, co ułatwia zrozumienie kodu oraz wygenerowanie dokumentacji do kodu.

Aplikacja pakowana jest do pliku .jar, co umożliwia uruchomienie jej na każdym komputerze z zainstalowaną maszyną wirtualną Javy w wersji 6.

3.7. Użytkowanie

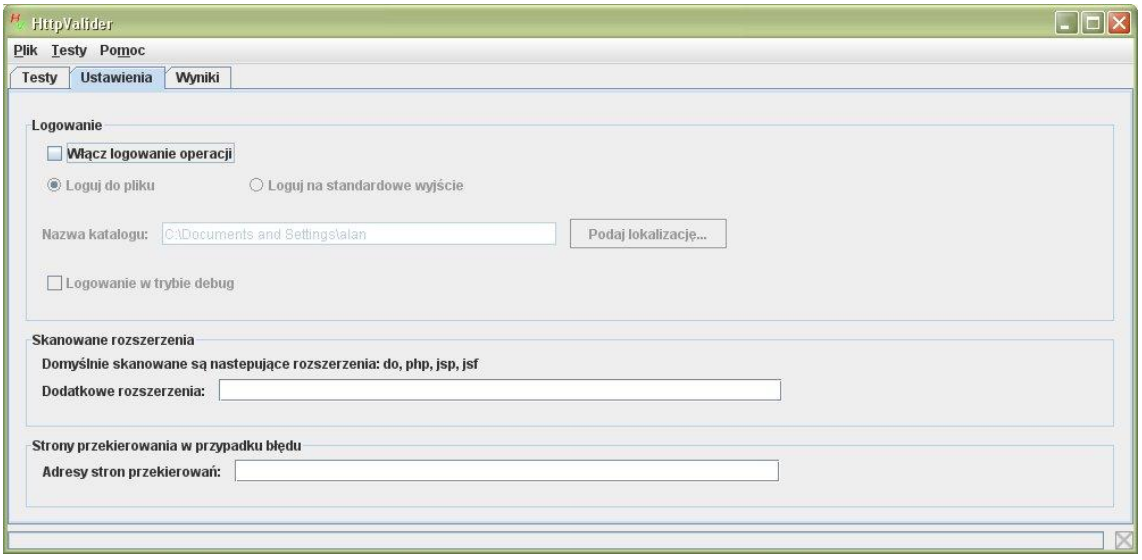
Po uruchomieniu aplikacji użytkownikowi ukazuje się jej główne okno, w którym na początku należy wybrać, czy dysponuje się plikiem z zapisanymi żądaniami wysłanymi i uzyskanymi wcześniej w trakcie komunikacji z testowaną aplikacją, czy zostanie podany jedynie adres do testowania. Następnie należy podać dane do testowania (w zależności od wybranej metody testowania) oraz wybrać testy bezpieczeństwa do przeprowadzenia. Wybrany przez użytkownika katalog zawierający pliki śladu zostanie zapisany w bazie danych znajdującej się w katalogu domowym użytkownika.

[22] Specyfikacja firmy Sun Microsystems określa JavaBeans jako: „wielokrotnie używalne komponenty, które mogą być manipulowane wizualnie przez narzędzia do tworzenia oprogramowania".



Rysunek 13. Główne okno aplikacji

Dodatkowo w zakładce „Ustawienia" użytkownik ma możliwość skonfigurowania aplikacji. Dostępne są dwa możliwe tryby logowania wykonywanych operacji, tryb „debug" powoduje zapis do logu pełnych śladów ze stosu (ang. *stacktrace*) w przypadku wystąpienia błędu w aplikacji. Wybrane ustawienia logowania zapisywane są w bazie danych aplikacji. Pole „skanowane rozszerzenia" pozwala na zdefiniowanie rozszerzeń plików, poza domyślnymi, które zostaną przetestowane. Żądania z pliku śladu odnoszące się do plików o innych rozszerzeniach (np. plików graficznych, plików stylu CSS) zostaną zignorowane. W polu „Strony przekierowania w przypadku błędu" HttpValider pozwala na zdefiniowanie stron, na które aplikacja przenosi użytkownika w przypadku wystąpienia błędu lub wykrycia niewłaściwych parametrów.



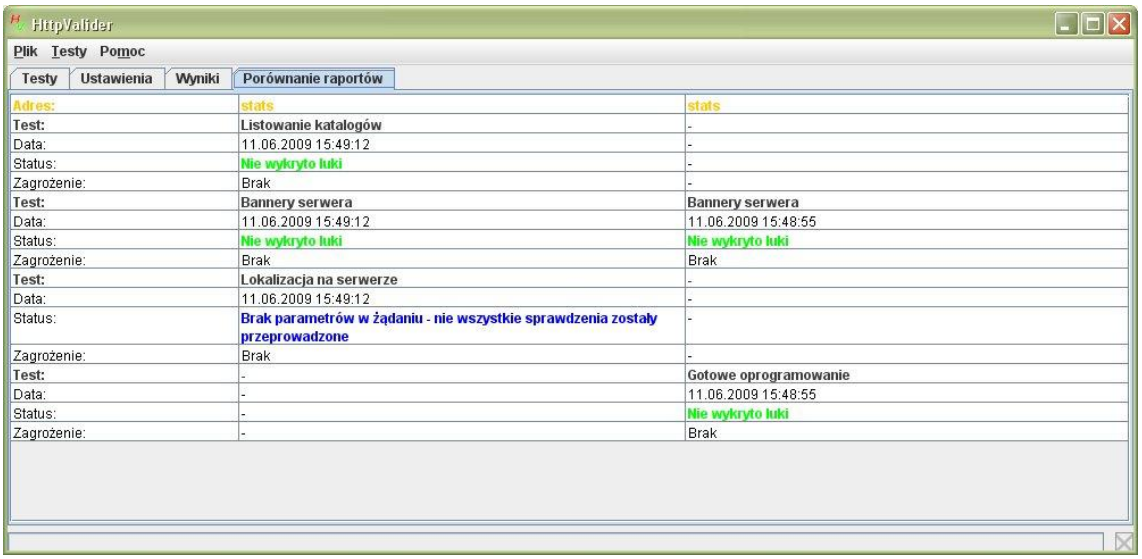
Rysunek 14. Okno ustawień aplikacji

Testowanie rozpoczyna się po wybraniu guzika „Start!”, zaś po zakończeniu testowania użytkownik jest przenoszony na kartę zawierającą podsumowanie dla wszystkich wykonanych testów oraz zakładki z wynikami poszczególnych testów.

Rysunek 15. Okno wyników testowania

Po zakończeniu testowania użytkownik może, wybierając odpowiednie opcje z menu „Testy”:

- Wyeksportować raport z testowania do formatu PDF. Użytkownikowi pokaże się okienko dialogowe z prośbą wyboru katalogu, gdzie zostanie zapisany raport.
- Zapisać raport z testowania do bazy danych. Użytkownikowi pokaże się okienko dialogowe informujące o sukcesie lub niepowodzeniu zapisu. Baza danych znajduje się w katalogu domowym użytkownika.
- Porównać raport z testowania z uprzednio wygenerowanym i zapisanym do bazy danych raportem. Użytkownikowi ukaże się okienko dialogowe z prośbą o wybór jednego z zapisanych w bazie danych raportów – ich identyfikatorami są daty i godziny wykonania sesji testowej. Po wybraniu jednego z nich użytkownik zostanie przeniesiony do nowej zakładki z wynikami porównania. W przypadku braku zapisanych raportów dla danej testowanej aplikacji użytkownikowi ukaże się okienko dialogowe ze stosownym komunikatem.



Rysunek 16. Okno porównania wyników testowania

Po zakończeniu testowania użytkownik może wybrać z menu „Plik” opcję „Nowy test” i rozpocząć kolejne testowanie. Możliwe jest również powrócenie do zakładki „Testy” i wykonanie kolejnego testowania.

4. Testowanie i walidacja aplikacji HttpValider

4.1. Przebieg testowania i działań walidacyjnych

Testowanie i walidację aplikacji HttpValider przeprowadzono w trzech etapach.

- W trakcie etapu pierwszego przetestowano wytworzony w pierwszym przyroście podsystem testów. Stworzono przypadki testowe używane do sprawdzenia poprawności przeprowadzanego przez aplikację procesu testowania. Celem etapu było wykazanie, że HttpValider odnajduje wszystkie stworzone w badanym kodzie luki i że nie dostarcza fałszywych wskazań, informując o nieistniejących lukach. Wszystkie środowiska zostały uruchomione lokalnie, bez udostępniania ich w Internecie.
- W trakcie etapu drugiego przetestowano funkcjonowanie całej aplikacji. Sprawdzono skuteczność testów na wdrożonych i użytkowanych aplikacjach webowych oraz porównano wyniki osiągane przez HttpValider z wybranymi innymi, dostępnymi za darmo, narzędziami. W trakcie całego etapu przeprowadzano testy regresyjne z etapu pierwszego. Celem tego etapu było wykazanie poprawności działania pełnej aplikacji.
- Etap trzeci polegał na przekazaniu HttpValidera użytkownikom zewnętrznym w celu oceny walorów użytkowych oraz stabilności aplikacji. Po otrzymaniu wyników pierwszego testowania znalezione błędy zostały naprawione, zaś HttpValider przekazany do drugiej tury testów. Etap umożliwił uczynienie interfejsu użytkownika bardziej przyjaznym, co zostało potwierdzone w trakcie drugiej tury testów.

4.2. Pierwszy etap testowania: środowiska testowe

W celu sprawdzenia poprawności przeprowadzanych testów wytworzono dwa proste środowiska testowe oraz skorzystano z gotowych aplikacji webowych, zawierające w swoim kodzie wszystkie badane luki z wyjątkiem luki dostępności ze względu na jej charakter wymagający dostępności aplikacji z Internetu. Każde ze środowisk składa się z kilku stron, z których tylko część zawiera luki.

Testy były prowadzone przez cały okres tworzenia aplikacji. Jej ostateczna wersja wykrywa wszystkie błędy środowisk testowych i w żadnym wypadku nie wskazuje nieistniejącej luki

4.2.1. Środowisko testowe dla aplikacji stworzonej w języku PHP

Tabela 52. Konfiguracja środowiska testowego dla aplikacji stworzonej w języku PHP

Język programowania	PHP 5.2.6
Baza danych	PostgreSQL 8.3.7, MySQL 5.0.75
Serwer WWW	Apache 2.2.11, lighttpd 1.4.19
System operacyjny	Linux Ubuntu 9.04
Występujące luki	Komunikaty i banery serwera Listowanie katalogów Ciasteczka Wstrzykiwanie kodu SQL Ukryte elementy formularza Dane o założonym zakresie wartości zwracanych Weryfikacja po stronie klienta Zmiana zachowania oprogramowania Lokalizacja oprogramowania na serwerze Ataki DOS

4.2.2. Środowisko testowe dla gotowych aplikacji napisanych w języku PHP

Tabela 53. Konfiguracja środowiska testowego dla gotowych aplikacji napisanych w języku PHP

Język programowania	PHP 5.2.6
Baza danych	MySQL 5.0.75
Serwer WWW	Apache 2.2.11
System operacyjny	Linux Ubuntu 9.04
Występujące luki	Gotowe oprogramowanie
Badane aplikacje	phpBB SMF WordPress Joomla

4.2.3. Środowisko testowe dla aplikacji stworzonej w języku Java

Tabela 54. Konfiguracja środowiska testowego dla aplikacji stworzonej w języku Java

Język programowania	Java 1.6.0
Baza danych	Brak
Serwer aplikacji	GlassFish 2.1
System operacyjny	Microsoft Windows XP SP3
Występujące luki	Komunikaty i banery serwera Przepełnienie bufora Weryfikacja po stronie klienta Zmiana zachowania oprogramowania Ataki DOS

4.3. Drugi etap testowania: testowanie aplikacji użytkowych i porównanie wyników z wynikami innych narzędzi

Do testowania zostały wybrane dwie aplikacje webowe użytkowane w Sieci Komputerowej Osiedla Studenckiego Politechniki Gdańskiej, działające na różnych fizycznych serwerach. W przypadku obu ich współautorem jest autor pracy, który jest także jednym z administratorów Sieci Komputerowej Osiedla Studenckiego Politechniki Gdańskiej.

Do testowania, oprócz aplikacji HttpValider, użyto:

- oprogramowania Nessus 2.2.10 (patrz: 2.2.1) do połączenia z którym użyto klienta OpenVAS-Client 1.0.4 do sprawdzenia, czy występują luki nie wymagające przechodzenia przez kolejne strony (wnioskowania),
- oprogramowania WebScarab (patrz: 2.2.9) do przetestowania, czy występują pozostałe luki.

4.3.1. System Rejestracji Użytkownika

System Rejestracji Użytkownika, dostępny pod adresem https://sru.ds.pg.gda.pl jest użytkowany w Sieci Komputerowej Osiedla Studenckiego Politechniki Gdańskiej do rejestracji i obsługi użytkowników sieci. Administratorzy mają dostęp do danych użytkowników, a także informacji statystycznych dotyczących sieci, np. wykorzystania puli adresów IP w poszczególnych domach studenckich Politechniki Gdańskiej. Rejestracja w systemie jest obowiązkowa dla użytkowników chcących korzystać z Internetu w domach studenckich Politechniki Gdańskiej, system zaś umożliwia im

aktualizację ich danych oraz zapoznanie się z informacjami dotyczącymi dostępu do Internetu.

System Rejestracji Użytkownika działa w środowisku o następujących parametrach:

Tabela 55. Konfiguracja środowiska Systemu Rejestracji Użytkownika

Język programowania	PHP 5.2.6
Baza danych	PostgreSQL 8.3.5
Serwer	Lighttpd 1.4.19
System operacyjny	Linux Debian 5.0

Dostęp do Systemu Rejestracji Użytkownika możliwy jest wyłącznie za pomocą protokołu HTTPS. Ponieważ certyfikat został podpisany przez Centrum Usług Informatycznych Politechniki Gdańskiej, które nie należy do grupy instytucji zaufania (ang. *Certificate Authority*), na czas testów został on wyłączony, zaś aplikacja udostępniona za pomocą protokołu HTTP.

Wyniki testowania przedstawiają się następująco:

Tabela 56. Wyniki testowania aplikacji System Rejestracji Użytkownika

Luka	**HttpValider**	**Nessus**	**WebScarab**
Konfiguracja serwera udostępniająca informacje przez HTTP			
Komunikaty i banery serwera	brak	brak	-
Listowanie katalogów	brak	brak	-
Nieoczekiwane dane wejściowe			
Przepełnienie bufora	brak	brak	-
Ciasteczka	brak	-	brak
Wstrzykiwanie kodu SQL	brak	-	brak
Ukryte elementy formularza	brak	-	brak
Dane o założonym zakresie wartości zwracanych	brak	-	brak
Weryfikacja po stronie klienta	brak	-	brak
Zmiana zachowania oprogramowania	występuje	-	brak
Hasła			
Zabezpieczenie hasłem	-	brak	brak
Siła haseł	-	brak	-
Inne			
Lokalizacja oprogramowania na serwerze	brak	brak	-
Ataki DOS	brak	-	brak
Gotowe oprogramowanie	brak	brak	-
Dostępność	występuje	-	-

Wyniki testowania aplikacją Nessus są zgodne z wynikami uzyskanymi z HttpValidera. Natomiast testowanie aplikacją WebScarab nie ujawniło podatności Systemu Rejestracji Użytkownika na atak XSS. Ponadto HttpValider wskazał fakt, że podsystem administratora Systemu Rejestracji Użytkownika dostępny jest w wyszukiwarce Google, co może ułatwić atak potencjalnemu atakującemu.

Podatność na atak XSS została już naprawiona – pole formularza rejestracji pozwalające na wpisanie loginu miało błędnie zaimplementowaną kontrolę wprowadzanych znaków. Atakujący mógł podać login np. będący skryptem napisanym w języku JavaScript, który zostawał zapisany w bazie danych i wywoływał się po wejściu administratora na profil tego użytkownika. Mogło to prowadzić np. do

przekierowania administratora na specjalnie spreparowana stronę www. Luka została usunięta przez poprawienie kontroli wprowadzanych znaków do pola loginu.

4.3.2. Strona domowa Domu Studenckiego nr 8

Strona domowa Domu Studenckiego nr 8 Politechniki Gdańskiej dostępna jest pod adresem http://ds8.ds.pg.gda.pl. Umożliwia ona mieszkańcom Domu Studenckiego nr 8 zapoznanie się z informacjami od Rady Mieszkańców, pobranie druków oraz zapisanie się na siłownię znajdującą się w Domu Studenckim nr 8. Pełny dostęp do aplikacji możliwy jest wyłącznie z adresów IP Domu Studenckiego nr 8, pozostali użytkownicy mają dostęp jedynie do krótkiej informacji na temat Domu Studenckiego nr 8.

Strona domowa Domu Studenckiego nr 8 działa w środowisku o następujących parametrach:

Tabela 57. Konfiguracja środowiska strony domowej Domu Studenckiego nr 8

Język programowania	PHP 5.1.2
Baza danych	MySQL 5.0.22
Serwer	Apache 2.0.55
System operacyjny	Solaris Nexenta 2.11

Wyniki testowania przedstawiają się następująco:

Tabela 58. Wyniki testowania strony domowej Domu Studenckiego nr 8

Luka	**HttpValider**	**Nessus**	**WebScarab**
Konfiguracja serwera udostępniająca informacje przez HTTP			
Komunikaty i banery serwera	brak	brak	-
Listowanie katalogów	brak	brak	-
Nieoczekiwane dane wejściowe			
Przepełnienie bufora	brak	brak	-
Ciasteczka	brak	-	brak
Wstrzykiwanie kodu SQL	brak	-	brak
Ukryte elementy formularza	brak	-	brak
Dane o założonym zakresie wartości zwracanych	brak	-	brak
Weryfikacja po stronie klienta	brak	-	brak
Zmiana zachowania oprogramowania	brak	możliwość ze względu na konfigurację serwera	brak
Hasła			
Zabezpieczenie hasłem	-	brak	brak
Siła haseł	-	brak	-
Inne			
Lokalizacja oprogramowania na serwerze	brak	brak	-
Ataki DOS	brak	-	brak
Gotowe oprogramowanie	brak	brak	-
Dostępność	brak	-	-

Testowanie wszystkimi aplikacjami dało zbliżone rezultaty – strona domowa Domu Studenckiego nr 8 nie jest podatna na żadne ataki. Co prawda oprogramowanie Nessus wskazało na możliwość ataku XSS, jednak wynika on z domyślnych ustawień serwera aplikacji umożliwiających badanie połączeń HTTP. Wykazane zagrożenie zostało określone jako średnie.

4.4. Trzeci etap testowania: opinie użytkowników zewnętrznych

HttpValider został przekazany użytkownikom zewnętrznym w celu oceny walorów użytkowych i stabilności. Testowanie to pozwoliło wprowadzić wiele poprawek w interfejsie graficznym użytkownika ułatwiających korzystanie z aplikacji. Uwagi były pomocne przy wprowadzaniu podpowiedzi objaśniających poszczególne funkcje oraz uzupełnianiu istniejących tekstów pomocy. Działanie to umożliwiło także poprawę stabilności aplikacji oraz ulepszenie jej wizualnej strony poprzez poprawki wprowadzone w GUI.

Aplikacja spotkała się z pozytywnym przyjęciem, została oceniona jako łatwa w obsłudze i intuicyjna. Wysoko została także oceniona jej przydatność dla twórców aplikacji webowych. Funkcje dodatkowe: generowanie raportów do formatu PDF oraz porównywanie wyników testowania przeprowadzonych w różnym czasie zostały ocenione jako pożyteczne i potrzebne.

4.5. Ograniczenia wytworzonej aplikacji

Przeprowadzone testy potwierdziły spełnienie przyjętych wymagań. Jednakże wytworzona aplikacja cechuje się następującymi ograniczeniami dotyczącymi procesu testowania bezpieczeństwa aplikacji webowych:

- Zakodowana na stałe baza sygnatur świadczących o lukach, co utrudnia jej ewentualną aktualizację. Uniemożliwia to użytkownikowi dodanie własnych sygnatur, co utrudnia użycie aplikacji do testowania nietypowych środowisk. Wada ta ogranicza także dostosowywanie HttpValidera do zmian technologicznych aplikacji webowych.
- Otrzymane odpowiedzi na żądania porównywane są jako całe strony internetowe. W przypadku dynamicznie generowanej zawartości strony internetowej HttpValider rozpozna otrzymane odpowiedzi jako różne strony (patrz: 3.3.2.2, 3.3.2.3). Częściowo udało się ominąć to ograniczenie poprzez możliwość zdefiniowania przez użytkownika domyślnych stron błędu i przekierowania.
- Brak możliwości przetestowania aplikacji, jeżeli używa ona nieznanego certyfikatu SSL, np. podpisanego przez niezaufaną instytucję (ang. *self-signed*).
- Operacja zapisu wyników testowania do bazy danych nie powoduje zapisania informacji dotyczących wykorzystanego przez użytkownika pliku śladu. Nie można zatem zobaczyć tych danych w trakcie porównywania wyników testowania.

- Aplikacje rozpoznawane są po nazwie domeny, zatem kilka różnych aplikacji działających w tej samej domenie zostanie oznaczonych tak samo, zaś wyniki ich testowania zostaną skojarzone w bazie danych.
- Do identyfikacji w bazie danych kolejnych sesji testowych danej aplikacji webowej używana jest data testowania. Użytkownik może wybrać do porównania wynik testowania wyłącznie po dacie przeprowadzenia sesji testowej.
- HttpValider nie testuje aplikacji webowych pod względem zabezpieczenia dostępu hasłem oraz mocy tych haseł.

Podsumowanie

Główne osiągnięcia pracy

Celem pracy było wytworzenie narzędzia wykrywającego luki bezpieczeństwa aplikacji webowych. Jednocześnie należało wykonać analizę metod ataków na aplikacje webowe poprzez protokół HTTP oraz opis dostępnych narzędzi i technik wykrywania luk bezpieczeństwa.

W niniejszej pracy przeanalizowano i sklasyfikowano występujące luki aplikacji webowych. Przeprowadzono pełny proces inżynierii oprogramowania, począwszy od zebrania wymagań na system, poprzez analizę, projekt, implementację, aż po testy. Stworzono narzędzie, które zostało ocenione przez użytkowników zewnętrznych jako bardziej funkcjonalne niż inne dostępne na rynku darmowe oprogramowanie w zakresie testowania aplikacji webowych poprzez protokół HTTP.

Stworzona aplikacja ma ponad 5000 linii kodu i pokrywa swoją funkcjonalnością wszystkie wymagania ze specyfikacji wymagań systemowych obejmujące trzynaście rodzajów luk bezpieczeństwa. Została także przetestowana na specjalnie przygotowanych środowiskach testowych, jak również na rzeczywistych systemach dostępnych w sieci Internet. Oprogramowanie zostało przetestowane także przez rzeczywistych użytkowników i uzyskało pozytywne recenzje. Porównanie skuteczności aplikacji ze skutecznością innych narzędzi wypadło dobrze (patrz: 4.3.1, 4.3.2), łatwość obsługi HttpValidera została oceniona wyższa niż porównywanych narzędzi, tj. Nessus i WebScarab.

Stworzenie aplikacji HttpValider było dla autora niniejszej pracy bardzo wartościowym doświadczeniem, ponieważ znacznie poszerzył swoją wiedzę zarówno dotyczącą ogólnie pojętego bezpieczeństwa w Internecie, jak również występujących luk w aplikacjach webowych. Poznał sposoby zapobiegania ich występowaniu oraz metody ich wykrywania. Dzięki aplikacji HttpValider udało się wykryć i naprawić luki w Systemie Rejestracji Użytkownika Sieci Komputerowej Osiedla Studenckiego Politechniki Gdańskiej.

Kierunki dalszego rozwoju

Najważniejszym kierunkiem w dalszym rozwoju aplikacji powinno być umożliwienie interakcji użytkownika z HttpValiderem. Ułatwi to obsługę sytuacji wymagających akcji użytkownika (np. podanie tokenu CAPTCHA). Pozwoli także na lepsze rozpoznanie sytuacji, kiedy otrzymane strony są identyczne, różnią się tylko dynamicznie generowanymi treściami (np. data, liczba użytkowników przeglądających

stronę). Da to również możliwość reakcji testera w przypadku, gdy wykonanie pewnych akcji możliwe jest tylko raz (np. rejestracja pod danym loginem).

Kolejnym kierunkiem rozwoju powinno być stworzenie zewnętrznej bazy sygnatur, co znacznie ułatwiłoby jej aktualizację i umożliwiłoby na przykład udostępnienie aktualizacji on-line. Tak stworzona baza sygnatur może stać się bazą wiedzy, skuteczniej analizującą otrzymane wyniki, niż obecnie zaimplementowany mechanizm wnioskujący.

W trakcie rozbudowy aplikacji można dodać testy niedotyczące protokołu HTTP. Pozwoliłoby to na rozszerzenie liczby odbiorców i na stworzenie kompletnego narzędzia testującego aplikacje webowe oraz ich środowisko.

Aplikacja dostępna jest wyłącznie w języku polskim, przetłumaczenie jej – w szczególności na język angielski – umożliwiłoby zwiększenie liczby potencjalnych odbiorców.

Wszystkie te działania będą prowadzić do ciągłej poprawy skuteczności testowania przez HttpValider oraz możliwie bliskiego kontaktu z użytkownikami w celu ciągłej aktualizacji aplikacji i szybkiego wprowadzania poprawek.

Bibliografia

1. **Wydawnictwo Naukowe PWN;** Internetowa Encyklopedia PWN. [Online] http://encyklopedia.pwn.pl/.

2. **Telekomunikacja Polska SA;** Słownik. [Online] [Zacytowano: 24 05 2009.] http://www.tp.pl/prt/pl/tpcert/bezpieczenstwo/slownik/?_a=664454.

3. **Główny Urząd Statystyczny;** *Społeczeństwo informacyjne: wykorzystanie technologii informacyjno-telekomunikacyjnych w 2006 r. (synteza).*

4. **Eurostat;** *UE: Internet w firmach 2008.*

5. *Europejska Konwencja w Sprawie Cyberprzestępczości.*

6. **Telekomunikacja Polska;** Klasyfikacja incydentów stosowana przez TP CERT. [Online] [Zacytowano: 04 03 2009.] http://www.tp.pl/prt/pl/tpcert/obsluga/zglaszanie/incydent/?_a=679496.

7. **CERT Polska;** *Analiza incydentów naruszających bezpieczeństwo teleinformatyczne zgłaszanych do zespołu CERT Polska w roku 2008.*

8. **SC Magazine UK;** *SC Magazine UK.* [Online] [Zacytowano: 10 12 2008.] www.scmagazineuk.com.

9. **Secunia;** Advisories by product. [Online] [Zacytowano: 05 03 2009.] http://secunia.com/advisories.

10. **Netcraft;** January 2009 Web Server Survey. [Online] [Zacytowano: 07 03 2009.] http://survey.netcraft.com/Reports/200901/.

11. **PandaLabs;** [Online] [Zacytowano: 07 03 2009.] www.pandalabs.com.

12. **Locos.pl;** *Bezpieczeństwo i audyt IT.* [Online] [Zacytowano: 31 01 2009.] www.locos.pl .

13. Powered By lighhtpd. [Online] [Zacytowano: 04 04 2009.] http://redmine.lighttpd.net/wiki/lighttpd/PoweredByLighttpd.

14. **Mirza Ahmad, D. R. i inni;** *Hack Proofing Your Network. Edycja Polska.* 2002.

15. **Dhanjani, N i Clarke, J.;** *Bezpieczeństwo sieci. Narzędzia.* Bytom : Helion, 2006.

16. **Schwartz, R. L. i Phoenix, T.;** *Wprowadzenie. Perl.* Gliwice : Helion, 2006.

17. **Cole, E., Krutz, R. L. i Conley, J.;** *Bezpieczeństwo sieci. Biblia.* Gliwice : Helion, 2005.

18. **Splaine, S.;** *Testing Web Security.* Indianapolis : Wiley, 2002.

19. **Podstawczyński, A.;** *Linux. Praktyczne rozwiązania.* Gliwice : Helion, 2000.

20. **O'Reilly;** What Is Web 2.0. [Online] [Zacytowano: 05 04 2009.] http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html.

21. **Bott, E. i Siechert, C.;** *Microsoft Windows. Bezpieczeństwo dla ekspertów.* Warszawa : RM, 2003.

22. **APOGEUM;** Plik robots.txt. [Online] [Zacytowano: 06 06 2009.] http://www.pozycjoner.org/artykuly/plik-robots-txt.html.

23. **Pierwszy Kongres Informatyki Polskiej;** *Raport: Strategia Rozwoju Informatyki w Polsce.* 1994.

Zawartość płyty CD

`\Praca magisterska`

`Alan Turower - Praca magisterska.docx` – praca dyplomowa w pliku MS Word 2007

`Alan Turower – Praca magisterska.pdf` – praca dyplomowa w pliku PDF

`\HttpValider`

`\zip` – plik wykonywalny aplikacji HttpValider spakowany do pliku zip

`\src` – źródła aplikacji HttpValider

`\Testy`

`\PHP` – środowisko testowe napisane w języku PHP

`\Java` – środowisko testowe napisane w języku Java