INŻYNIERIA OPROGRAMOWANIA W PROCESACH INTEGRACJI SYSTEMÓW INFORMATYCZNYCH
Pod redakcją J. Górskiego, C. Orłowskiego, 2010 PWNT Gdańsk

AKTYWNA METODA TESTOWANIA BEZPIECZEŃSTWA APLIKACJI WEBOWYCH HTTPVALIDER ORAZ OCENA JEJ SKUTECZNOŚCI

Alan TUROWER[*], Andrzej WARDZIŃSKI[*]

1. Wprowadzenie

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. 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. Powoduje to konieczność częstego i skutecznego testowania aplikacji webowych [11].

W rozdziale przedstawiona jest metoda HttpValider [12] aktywnego testowania bezpieczeństwa aplikacji webowych oraz ocena jej skuteczności. Omówiono zagadnienia oceny bezpieczeństwa, w szczególności podejście pasywnego i aktywnego testowania bezpieczeństwa aplikacji webowych. Przedstawiono kroki metody HttpValider oraz omówiono narzędzie implementujące opracowaną metodę. Przedstawiono wyniki testów porównawczych narzędzia HttpValider z dwoma innymi narzędziami.

2. Testowanie bezpieczeństwa aplikacji webowych

Celem działań zapewnienia bezpieczeństwa aplikacji webowych jest uniemożliwienie wykonania skutecznego ataku na system lub ograniczenie jego skutków.

[*] Politechnika Gdańska, Wydział Elektroniki Telekomunikacji i Informatyki, Katedra Inżynierii Oprogramowania, e-mail: alan.turower@eti.pg.gda.pl, andrzej.wardzinski@eti.pg.gda.pl.

Atak jest działaniem mającym na celu włamanie się do aplikacji webowej, zmianę sposobu jej działania lub zablokowanie dostępu do niej [3].

Włamanie jest to bezprawne uzyskanie dostępu do systemu lub sieci komputerowej mające miejsce po pokonaniu zabezpieczeń lub wykorzystaniu podatności oprogramowania zainstalowanego na komputerze lub podatności sieci [3].

Luka bezpieczeństwa jest słabością aplikacji pozwalającą atakującemu na włamanie [3].

Ocena bezpieczeństwa jest procesem mającym na celu określenie podatności systemu na ataki oraz potencjalne ich konsekwencje. Jednym z głównych elementów oceny bezpieczeństwa jest identyfikacja luk bezpieczeństwa badanej aplikacji [3]. Zazwyczaj jest przeprowadzana w jednym z następujących celi:

- testowanie aplikacji webowej przed jej wdrożeniem lub przed wdrożeniem jej nowej wersji;
- monitorowanie aplikacji webowej w trakcie jej działania;
- wykrywanie słabości aplikacji webowej w trakcie przygotowywania ataku.

W rozdziale skupimy się na pierwszym elemencie oceny bezpieczeństwa – testowaniu aplikacji webowej. Celem testowania aplikacji webowej jest wykrycie występujących w niej luk bezpieczeństwa przed wdrożeniem aplikacji oraz przed wdrożeniem jej nowej wersji, w celu upewnienia się, że dokonane zmiany nie wprowadziły luk bezpieczeństwa [4].

Testy bezpieczeństwa aplikacji webowych mogą być przeprowadzone stosując dwa podejścia: pasywne lub aktywne. Testowanie pasywne polega na obserwowaniu ruchu pomiędzy aplikacją i jej klientami oraz analizie przesyłanych danych w celu identyfikacji luk bezpieczeństwa [8]. Takie testy mogą być wykonane dla normalnego ruchu wywołanego przez użytkowników testowych. Można też zaangażować osoby potrafiące przygotować ataki do użycia testowej aplikacji do generowania odpowiedniego ruchu w sieci. Sama metoda testów pasywnych obejmuje tylko obserwowanie przesyłanych komunikatów i ich analizę, nie obejmuje zarządzania zakresem testów i atakami testowymi. Głównymi wadami podejścia jest wysokie ryzyko, że [10]:

- nie wszystkie ścieżki testowe (np. formatki, formularze) zostaną pokryte w czasie testów,
- nie wszystkie możliwe formy ataku zostaną wykorzystane dla miejsc aplikacji podatnych na dany rodzaj ataku, wobec czego nie wszystkie luki zostaną wykryte,

Testowanie aktywne zakłada, że dla każdej testowanej strony aplikacji webowej zostają opracowane warianty kolejnych zapytań, które są przygotowane w taki sposób, aby sprawdzić istnienie luki bezpieczeństwa. Na przykład jeżeli na stronie aplikacji webowej znajduje się formularz z polem tekstowym, to dla testu odporności na atak wstrzykiwania kodu SQL (*SQL injection*) działanie testera polega na przygotowaniu zapytań, w których w wysyłanym polu formularza znajduje się kod mogący spowodować wykonanie skryptu SQL po stronie serwera. Każde przygotowane zapytanie stanowi osobny przypadek testowy i wymaga wysłania do serwera w kontekście danej strony aplikacji. W odpowiedzi na zapytanie serwer aplikacji wysyła odpowiedź, która jest analizowana przez testera. Zwykle zakłada się, że skuteczny atak powoduje zmianę odpowiedzi serwera w porównaniu do poprawnego zapytania i tester może porównać znaną mu poprawną odpowiedź dla danej strony aplikacji z odpowiedzią uzyskaną dla zapytania zawierającego próbę ataku. Dla aplikacji odpornej na atak odpowiedź powinna być albo zgodna z normalną odpowiedzią albo zawierać komunikat o wykryciu próby ataku. Inne odpowiedzi mogą sugerować istnienie luki bezpieczeństwa. Do testera należy ocena, czy uzyskany wynik pozwala już na stwierdzenie obecności danej luki czy wymagane są dalsze testy oraz czy dotyczy jednej luki czy kilku powiązanych luk [5].

Aktywne metody w porównaniu do metod pasywnych zawierają działania planowania testów, co umożliwia lepsze zarządzanie zakresem testów i zwykle większą skuteczność, ale wymagają większego zaangażowania testera oraz istnieje możliwość niepoprawnej interpretacji otrzymanych danych, na przykład odpowiedzi JSON lub przepływu zdarzeń w interfejsie użytkownika. [7] Testowanie aktywne jest bardziej skomplikowane w przypadku aplikacji

działającej na zasadzie sieci komponentów [1]. Problemem obu podejść jest sposób analizy odpowiedzi przesyłanych przez serwer i diagnozowanie występowania luk bezpieczeństwa.

Bardziej zaawansowane metody testowania, zarówno pasywne jak i aktywne, mogą zakładać dostęp testera do bazy danych oraz plików serwera, co daje testerowi większe możliwości diagnozowania skuteczności ataku.

Podczas testowania należy też być świadomym, że wszystkie działania testów bezpieczeństwa mogą spowodować w wypadku skutecznego ataku chwilowe lub trwałe uszkodzenie aplikacji lub serwera. Wprawdzie przypadki testowe są zwykle tak dobierane, aby wykazać istnienie luki, ale nie powodować istotnych uszkodzeń, tym niemniej testy należy przeprowadzać na kopii systemu, a nie na systemie eksploatowanym przez użytkowników.

Jedynym podejściem oceny bezpieczeństwa, który nie powoduje zagrożenia uszkodzenia serwera jest przegląd kodu pod kątem sprawdzenia zabezpieczeń dla poszczególnych luk.

3. Metoda aktywna HttpValider

W ramach prac badawczo-rozwojowych Katedry Inżynierii Oprogramowania Politechniki Gdańskiej opracowano metodę HttpValider realizującą aktywne podejście prowadzenia testów bezpieczeństwa aplikacji webowych oraz zaprojektowano i wytworzono narzędzie wspierające testy [10]. Metoda pozwala na wykrycie luk bezpieczeństwa wymienionych w tabeli 1. Luki te zostały opracowane na podstawie raportu CERT Polska oraz innych źródeł [2] [9] [12].

Tab. 1. Luki bezpieczeństwa wykrywane przez metodę HttpValider

Konfiguracja serwera udostępniająca informacje przez HTTP
Obecność komunikatów i banerów serwera
Możliwość listowania katalogów
Nieoczekiwane dane wejściowe
Przepełnienie bufora
Zatrucie ciasteczek
Wstrzykiwanie kodu SQL
Zatrucie ukrytych elementów formularza
Zatrucie danych o założonym zakresie wartości zwracanych
Weryfikacja danych występująca wyłącznie po stronie klienta
Wstrzyknięcie złośliwego kodu (Cross-site scripting)
Inne
Lokalizacja oprogramowania na serwerze pozwalająca na dostęp do zawartości serwera
Ataki DOS (Denial of Service)
Informacja o użyciu gotowego oprogramowania
Dostępność stron administracyjnych w wyszukiwarkach internetowych

3.1. Kroki stosowania metody HttpValider

Zastosowanie metody odbywa się w następujących krokach:

1. **Przygotowanie środowiska** – Tester przygotowuje kopię środowiska i testowanej aplikacji, aby zapobiec uszkodzeniu systemu rzeczywistego. Określa także zakres testów: rodzaje luk bezpieczeństwa oraz ścieżki użycia aplikacji (formatki) podlegających testom.
2. **Przygotowanie pliku śladu** – Stosując przeglądarkę z dodatkiem zapewniającym rejestrowanie interakcji (nazywanej dalej śladem) Tester przechodzi przez aplikację webową, wywołując wszystkie możliwe zdarzenia w zakresie ścieżki testowej. Zarejestrowany w pliku ślad obejmuje listę żądań HTTP: GET oraz POST wraz ze wszystkimi ich parametrami.
3. **Modyfikacje żądań** – W każdym kolejnym, jeszcze nie modyfikowanym, żądaniu z pliku śladu, narzędzie dodaje, modyfikuje lub usuwa parametry, w sposób zdefiniowany dla

testowanej luki bezpieczeństwa w zakresie testów. Stosowane są następujące schematy tworzenia żądań dla poszczególnych rodzajów luk:

a. **Możliwość listowania katalogów**: wywołanie żądań GET wyświetlenia kolejnych katalogów znajdujących się w ścieżce oryginalnego żądania.
b. **Przepełnienie bufora**: zastąpienie kolejno oryginalnych parametrów żądania znaczną ilością danych (np. poprzez wielokrotne powtórzenie wartości parametru).
c. **Zatrucie ciasteczek**: przesłanie zamiast oryginalnej zawartości ciasteczka znaków specjalnych, treści niezgodnej z logiką aplikacji lub znacznej ilości tekstu (próba przepełnienia bufora poprzez ciasteczka).
d. **Wstrzykiwanie kodu SQL**: modyfikacja parametru, który może być przekazany przez aplikację do bazy danych na zawierający specjalne znaki formatujące. Dla bazy MySQL taki ciąg może zawierać łańcuch znaków „`x' OR '1'='1`".
e. **Zatrucie ukrytych elementów formularza**: przesłanie zamiast oryginalnej wartości ukrytego pola formularza znaków specjalnych, treści niezgodnej z logiką aplikacji lub znacznej ilości tekstu (próba przepełnienia bufora poprzez ukryte pole formularza).
f. **Zatrucie danych o założonym zakresie wartości zwracanych**: przesłanie zamiast oryginalnej wartości innej, niewystępującej w prawdopodobnym zbiorze możliwych wartości zwracanych (np. poprzez zastąpienie wartości liczbowej ciągiem znaków).
g. **Weryfikacja danych występująca wyłącznie po stronie klienta**: w przypadku wykrycia weryfikacji danych występującej po stronie klienta przesłanie zamiast oryginalnej wartości pola formularza znaków specjalnych, treści niezgodnej z logiką aplikacji lub znacznej ilości tekstu (próba przepełnienia bufora poprzez pole formularza weryfikowane wyłącznie po stronie klienta).
h. **Wstrzyknięcie złośliwego kodu** (*Cross-site scripting*): przesłanie zamiast oryginalnego parametru prostego kodu w języku wykonywalnym po stronie przeglądarki, np. JavaScript.
i. **Lokalizacja oprogramowania na serwerze pozwalająca na dostęp do zawartości serwera**: w przypadku wykrycia parametru, którym aplikacja może odwoływać się do pliku, zmiana potencjalnej ścieżki do pliku na bliższą korzeniowi serwera, np., poprzez dodanie do wartości parametru „`../`".
j. **Ataki DOS** (*Denial of Service*): wielokrotne wysłanie żądania zawierającego znaczną ilość danych w krótkim czasie.
k. **Dostępność stron administracyjnych w wyszukiwarkach internetowych**: wysłanie do wyszukiwarki internetowej zapytania o dostępność stron administracyjnych. Dla wyszukiwarki Google takie zapytanie może zawierać parametry: `site:www.aplikacja.pl/admin +filetype:*`.

4. **Wysłanie żądania** – Każde wygenerowane żądanie jest wysyłane do aplikacji, a następnie odbierana jest odpowiedź. Żądanie powinno zawierać informacje o sesji otrzymane w trakcie procesu testowania, o ile występują.
5. **Porównanie odpowiedzi z bazą sygnatur** – Odpowiedzi na żądanie porównywane są z bazą sygnatur. Baza sygnatur zawiera elementy zdefiniowane osobno dla każdej z poszukiwanych luk bezpieczeństwa, których obecność w odpowiedzi na żądanie świadczy o występowaniu danej luki. Jeżeli odpowiedź zawiera element z bazy sygnatur, zaś oryginalna odpowiedź, zarejestrowana w kroku 2, nie zawiera jej, zwracana jest informacja o wystąpieniu luki bezpieczeństwa.
6. **Zwrócenie wyniku testu** – Po wysłaniu wszystkich zmodyfikowanych żądań i przeanalizowaniu otrzymanych odpowiedzi zwracany jest wynik dla testu bezpieczeństwa.

3.2. Projektowanie narzędzia wspierającego metodę

Dla przedstawionej metody zaprojektowano i zaimplementowano narzędzie wykonujące automatycznie testy bezpieczeństwa. Do implementacji wybrano język Java 6. Wykorzystano środowisko NetBeans, biblioteki Swing, iText, Apache Derby (JavaDB) [12].

Podczas projektowania narzędzia wykonano optymalizację algorytmu, dzięki której jest możliwe zminimalizowanie liczby wysyłanych zapytań HTTP. Zbyt duża ich liczba może spowodować alarm bezpieczeństwa w sieci lub samej aplikacji webowej, co może zmienić zachowanie się testowanej aplikacji. Podczas optymalizacji stosowano następujące zasady:

- Wysyłanie zmodyfikowanych żądań dla danego żądania pierwotnego jest przerywane po wykryciu luki bezpieczeństwa.
- Niektóre testy wykonywane są tylko raz, niezależnie od liczby żądań pierwotnych zarejestrowanych w pliku śladu, jeżeli jedno żądanie wystarczy do wykrycia luki.
- Niektóre żądania testowe mogą służyć do wykrycia więcej niż jednej luki. Na przykład występowanie podatności na wstrzykiwanie kodu SQL często świadczy o weryfikacji danych wyłącznie po stronie klienta. Wykrycie takiej podatności powoduje zgłoszenie występowania kilku luk.

Opracowane narzędzie pozwala na efektywne prowadzenie testów bezpieczeństwa aplikacji webowych, przy czym charakteryzuje się następującymi ograniczeniami:

Kompletność testowania: Dla zapewnienia kompletności testowania aplikacji webowej wymagana jest jej przejście przez Testera przez wszystkie możliwe strony aplikacji oraz wykonanie na nich wszystkich możliwych akcji oraz wypełnienie wszystkich możliwych formularzy. Jest to konieczne, by zweryfikować wszystkie ścieżki działania aplikacji.

Dodatkowo aplikacja webowa może przyjmować parametry, których nie można przekazać z poziomu interfejsu użytkownika aplikacji i nie są udokumentowane, jednak aplikacja obsługuje je poprawnie – tzw. tylne drzwi (ang. *backdoor*) [3]. Bez wiedzy o takich parametrach nie jest możliwe przetestowanie obsługi przyjmowanych przez nie wartości.

Dynamiczna zawartość stron aplikacji: Testowana aplikacja webowa może nakładać ograniczenia na przyjmowane parametry, np. nie pozwalać na wielokrotne utworzenie dokumentu lub konta w systemie. Metoda HttpValider w czasie testowania może wielokrotnie wysłać żądanie zawierające te same parametry, jednak zostanie ono obsłużone za drugim i kolejnymi razami inaczej niż pierwsze, powodując wyświetlenie błędnego wyniku testu, np. w przypadku sprawdzenia poprawności dodawania nowego dokumentu.

Niepowtarzalne akcje 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 metodą, a w niektórych wypadkach wyłączenia zabezpieczeń uniemożliwiających automatyczne testowanie aplikacji.

Dostęp do Internetu: testy możliwości listowania katalogów oraz dostępności w Google wymagają dostępu narzędzia do Internetu.

4. Ocena skuteczności metody

Opracowane narzędzie HttpValider [12] zostało poddane ocenie skuteczności testów bezpieczeństwa. Ocena została przeprowadzona w dwóch etapach:

1. walidacja narzędzia – celem tego etapu była ocena, czy metoda pozwala na skuteczne wykrycie luk bezpieczeństwa określonych w tabeli 1.
2. testów porównawcze metodą HttpValider oraz innymi dostępnymi narzędziami i porównanie wyników; przedmiotem testów były dwie aplikacje wdrożone i stosowane na Politechnice Gdańskiej.

4.1. Weryfikacja skuteczności testów

Przygotowanie weryfikacji obejmowało wytworzenie dwóch prostych środowisk testowych oraz skorzystanie z gotowych aplikacji webowych, zawierających 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. Ma to na celu sprawdzenie, czy metoda nie tylko wykrywa luki bezpieczeństwa, lecz także czy nie zgłasza tzw. „fałszywych alarmów" informując o nieistniejących zagrożeniach.

Testy przeprowadzono na następujących konfiguracjach:

- Aplikacja w języku PHP działająca na serwerze Apache z bazą danych PostgreSQL w systemie operacyjnym Linux Ubuntu;
- Aplikacja w języku PHP działająca na serwerze lighttpd z bazą danych MySQL w systemie operacyjnym Linux Ubuntu;
- Aplikacja w języku Java działająca na serwerze GlassFish w systemie operacyjnym Microsoft Windows XP.

Metoda wykryła wszystkie istniejące luki w poszczególnych środowiskach, nie zgłaszając przy tym ani jednego fałszywego alarmu. Potwierdziło to skuteczność metody HttpValider w zakresie luk bezpieczeństwa określonych w tabeli 1.

4.2. Porównanie skuteczności testów

Skuteczność metody HttpValider została oceniona poprzez porównanie z innymi darmowymi narzędziami stosującymi aktywną metodę testów bezpieczeństwa aplikacji webowych [6]. Wybrano dwa takie narzędzia, najpełniej pokrywające zakres testów pokrywany przez metodę HttpValider:

- Nessus do połączenia z którym użyto klienta OpenVAS-Client do sprawdzenia, czy występują luki nie wymagające przechodzenia przez kolejne strony (wykrywania możliwości wystąpienia luki na stronie na podstawie zawartości poprzedniej);
- WebScarab do przetestowania, czy występują pozostałe luki.

Przedmiotem testów porównawczych były dwie aplikacje webowe użytkowane w Sieci Komputerowej Osiedla Studenckiego Politechniki Gdańskiej. Aplikacja A funkcjonuje od 2 lat i jest używana przez 3 000 użytkowników, zaś aplikacja B jest stosowana od roku i jest używana przez 300 użytkowników. Testowane aplikacje działają na następujących konfiguracjach systemowych:

- aplikacja A w języku PHP działająca na serwerze lighttpd z bazą danych PostgreSQL w systemie operacyjnym Linux Debian 5.0;
- aplikacja B w języku PHP działająca na serwerze Apache z bazą danych MySQL w systemie operacyjnym Solaris Nexenta.

Wyniki testowania przedstawiono w tabelach 2 i 3. Słowo *brak* oznacza, że aplikacja nie stwierdziła istnienia luki, *występuje* oznacza, że aplikacja wykryła lukę, zaś myślnik oznacza, że aplikacja nie była użyta do testowania występowania danej luki.

Tab. 2. Wyniki testowania aplikacji A

Luka	HttpValider	Nessus	WebScarab
Konfiguracja serwera udostępniająca informacje przez HTTP			
Obecność komunikatów i banerów serwera	brak	brak	-
Możliwość listowania katalogów	brak	brak	-
Nieoczekiwane dane wejściowe			
Przepełnienie bufora	brak	brak	-
Zatrucie ciasteczek	brak	-	brak

Wstrzykiwanie kodu SQL	brak	-	brak
Zatrucie ukrytych elementów formularza	brak	-	brak
Zatrucie danych o założonym zakresie wartości zwracanych	brak	-	brak
Weryfikacja danych występująca wyłącznie po stronie klienta	brak	-	brak
Wstrzyknięcie złośliwego kodu (Cross-site scripting)	występuje	-	brak
Inne			
Lokalizacja oprogramowania na serwerze pozwalająca na dostęp do zawartości serwera	brak	brak	-
Ataki DOS (Denial of Service)	brak	-	brak
Informacja o użyciu gotowego oprogramowania	brak	brak	-
Dostępność stron administracyjnych w wyszukiwarkach internetowych	występuje	-	-

Wyniki testowania narzędziem Nessus są zgodne z wynikami uzyskanymi z HttpValidera. Natomiast testowanie narzędziem WebScarab nie ujawniło podatności aplikacji A na atak *Cross-site scripting*. Ponadto HttpValider wskazał fakt, że podsystem administratora aplikacji A dostępny jest w wyszukiwarce Google, co może ułatwić atak potencjalnemu atakującemu.

Analiza aplikacji webowej po testach wykazała istnienie dwóch luk wykrytych przez HttpValider, w wyniku czego administratorzy usunęli wskazane luki.

Tab. 3. Wyniki testowania aplikacji B

Luka	**HttpValider**	**Nessus**	**WebScarab**
Konfiguracja serwera udostępniająca informacje przez HTTP			
Obecność komunikatów i banerów serwera	brak	brak	-
Możliwość listowania katalogów	brak	brak	-
Nieoczekiwane dane wejściowe			
Przepełnienie bufora	brak	brak	-
Zatrucie ciasteczek	brak	-	brak
Wstrzykiwanie kodu SQL	brak	-	brak
Zatrucie ukrytych elementów formularza	brak	-	brak
Zatrucie danych o założonym zakresie wartości zwracanych	brak	-	brak
Weryfikacja danych występująca wyłącznie po stronie klienta	brak	-	brak
Wstrzyknięcie złośliwego kodu (Cross-site scripting)	brak	brak	brak
Inne			
Lokalizacja oprogramowania na serwerze pozwalająca na dostęp do zawartości serwera	brak	brak	-
Ataki DOS (Denial of Service)	brak	-	brak
Informacja o użyciu gotowego oprogramowania	brak	brak	-
Dostępność stron administracyjnych w wyszukiwarkach internetowych	brak	-	-

Testowanie aplikacji B wszystkimi narzędziami dało identyczne rezultaty – aplikacja ta nie jest podatna na żadne ataki w określonym zakresie luk.

Dla testowanych dwóch aplikacji webowych narzędzie HttpValider wykazało się większym zakresem testów niż narzędzia Nessus i WebScarab oraz wyższą skutecznością.

HttpValider oraz WebScarab były znacznie prostsze w użyciu niż Nessus, jednakże więcej czasu zajmuje w nich przygotowanie testów oraz interpretacja wyników. Wszystkie narzędzia charakteryzują się podobnym czasem przeprowadzenia testów.

5. Podsumowanie

W rozdziale przedstawiono metody pasywne i aktywne testowania bezpieczeństwa aplikacji webowych. Zaproponowano algorytm HttpValider testowania metodą aktywną dla zakresu obejmującego 13 luk bezpieczeństwa. Dla przedstawionej metody zaprojektowano i zaimplementowano narzędzie je wspierające. Następnie porównano skuteczność metody ze skutecznością innych dostępnych rozwiązań. Testy porównawcze wykazały, że opracowane narzędzie pozwala na skuteczne i efektywne prowadzenie testów bezpieczeństwa aplikacji webowych.

Bibliografia

[1] Cavalli A.R., Benameur A., Mallouli W., Li K.: *A Passive Testing Approach for Security Checking and its Practical Usage for Web Services Monitoring*, NOTERE, 2009.
[2] CERT Polska: *Analiza incydentów naruszających bezpieczeństwo teleinformatyczne zgłaszanych do zespołu CERT Polska w roku 2008*, www.cert.pl/PDF/Raport_CP_2008.pdf (28.03.2010)
[3] Cole, E., Krutz, R. L., Conley, J.: *Bezpieczeństwo sieci. Biblia*, Helion, 2005.
[4] Curphey M., Araujo R.: *Web Application Security Assessment Tools*, IEEE Explore, 2006.
[5] Fong E., Okun V.: *Web Application Scanners: Definitions and Functions*, IEEE Explore, 2007.
[6] Forum of Computer Engineers: *Security Analysis & Hacking*, National Institute of Technology, Hamirpur, 2010.
[7] Zaleski M.: *ratproxy: passive web application security assessment tool*, http://code.google.com/p/ratproxy/ (25.03.1010).
[8] Lee D. and others: *Passive testing and application s to network management*, ICNP, 1997.
[9] Russel. R. and others: *Hack Proofing Your Network. Edycja Polska,* Helion, 2002.
[10] Splaine, S.: *Testing Web Security*, Wiley, 2002.
[11] Suto L.: *Analyzing the Accuracy and Time Costs of Web Application Security Scanners*, ha.ckers, 2009.
[12] Turower A.: *System wykrywania luk bezpieczeństwa aplikacji webowych*, Politechnika Gdańska, 2009.

{Odrębna strona streszczeń i adresów}

Aktywna metoda testowania bezpieczeństwa aplikacji webowych HttpValider oraz ocena jej skuteczności

A. Turower, A. Wardziński

Politechnika Gdańska
Wydział Elektroniki, Telekomunikacji i Informatyki
Katedra Inżynierii Oprogramowania
alan.turower@eti.pg.gda.pl, andrzej.wardzinski@eti.pg.gda.pl e-mail

Rozdział 5. Aktywna metoda testowania bezpieczeństwa aplikacji webowych HttpValider oraz ocena jej skuteczności. W rozdziale omówiono zagadnienia oceny bezpieczeństwa aplikacji webowych, w szczególności podejście pasywnego i aktywnego testowania bezpieczeństwa. Przedstawiono metodę aktywnego testowania HttpValider obejmującą generowanie żądań HTTP pozwalające na identyfikację 13 luk bezpieczeństwa. Dla opracowanej metody zaprojektowano i zaimplementowano narzędzie wspierające automatyczne przeprowadzenie testów dla scenariusza użycia testowanej aplikacji zarejestrowane w pliku śladu przez testera. Skuteczność metody została potwierdzona podczas walidacji wykonanej dla testowych aplikacji zawierających oraz nie zawierających poszczególne luki bezpieczeństwa. Wykonano również testy porównawcze HttpValidera z dwoma innymi narzędziami testów bezpieczeństwa dla dwóch aplikacji webowych. Przedstawiono wyniki testów potwierdzające skuteczność metody HttpValider w wykrywaniu zbioru 13 luk bezpieczeństwa.

Web Application Security Active Testing Method HttpValider and its Effectiveness Assessment

A. Turower, A. Wardziński

Gdańsk University of Technology
Faculty of Electronics, Telecommunications and Informatics
Department of Software Engineering
alan.turower@eti.pg.gda.pl, andrzej.wardzinski@eti.pg.gda.pl

Chapter 5. Web Application Security Active Testing Method HttpValider and its Effectiveness Assessment. The chapter concerns web application security assessment, in particular passive and active approach to security testing. Active testing HttpValider method is proposed. It uses HTTP request generation for detecting 13 types of security vulnerabilities. A software tool was designed and implemented. The tool performs automatic tests for a web application usage scenario recorded by the tester. The method effectiveness was proved by the validation conducted for a set of test web applications with and without any security vulnerabilities. A test has been conducted to compare Httpvalider effectiveness with two other security testing tools. The comparative test was performed for two web applications. The results presented in the chapter confirm HttpValider effectiveness for detecting a set of 13 security vulnerabilities.