PRZEGLĄD TELEINFORMATYCZNY
NR 2, 2013

Metodyki testowania bezpieczeństwa aplikacji internetowych

Marek ANTCZAK, Zbigniew ŚWIERCZYŃSKI

Instytut Teleinformatyki i Automatyki WAT,
ul. Gen. S. Kaliskiego 2, 00-908 Warszawa
marek.antczak@gmail.com, z.swierczynski@ita.wat.edu.pl

STRESZCZENIE: Artykuł stanowi wprowadzenie do problematyki testów bezpieczeństwa aplikacji internetowych. Przedstawiono w nim główne cele stawiane testom bezpieczeństwa tego typu aplikacji. Zasygnalizowano obszary związane z najczęściej występującymi podatnościami aplikacji WWW oraz ogólne sposoby ich identyfikacji. Dokonano ogólnego przeglądu oraz próby porównania dostępnych metodyk testowania bezpieczeństwa aplikacji internetowych. Przy tej okazji wskazano niedoskonałości istniejących metodyk, tym samym uzasadniając celowość dalszych prac w zakresie opracowania spójnej i kompletnej metodyki testowania aplikacji internetowej wspieranej przez odpowiednio dobrane narzędzia aplikacyjne.

SŁOWA KLUCZOWE: bezpieczeństwo aplikacji internetowych, testy bezpieczeństwa, błędy bezpieczeństwa, metodyka testów bezpieczeństwa, testy penetracyjne, atrybuty bezpieczeństwa

1. Wprowadzenie

W dobie powszechnej informatyzacji różnorodnych usług oferowanych za pośrednictwem sieci Internet zarówno instytucje rządowe, przedsiębiorstwa komercyjne, jak i prywatni obywatele, muszą troszczyć się o swoje zasoby informacyjne udostępniane z wykorzystaniem tzw. usług WWW[1] w kontekście: integralności, poufności i dostępności. Istotność tego zagadnienia rośnie, ponieważ poprzez serwisy internetowe coraz częściej realizowany jest dostęp do zaawansowanych usług i systemów. Przykładem mogą być serwisy udostępniające usługi pocztowe (ang. *webmail*), serwisy aukcyjne (np. *e-bay*,

[1] Usługi WWW, usługi webowe, serwisy internetowe, portale internetowe i witryny internetowe to inne określenia dla aplikacji internetowych.

Allegro), serwisy społecznościowe (np. *Facebook*), e-banking (np. *mBank*), czy inne różnego rodzaju płatności internetowe (np. *PayPal*, *Dotpay*). W związku z rosnącym zakresem usług świadczonych przez aplikacje internetowe nieprawidłowa ich implementacja, o co nie trudno przy wzrastającej ich złożoności, wiąże się z coraz większym ryzykiem i bardziej odczuwalnymi różnego rodzaju stratami. O znaczeniu tych zagrożeń świadczy raport oceniający bezpieczeństwo tego typu usług w II kwartale 2012 roku opublikowany przez Rządowy Zespół Reagowania na Incydenty Komputerowe CERT.GOV.PL [16]. Stwierdza się w nim, że w okresie objętym raportem w różnych witrynach internetowych należących do instytucji państwowych znaleziono 213 błędów, z czego: 31 o poziomie zagrożenia bardzo wysokim, 45 błędów o poziomie wysokim, 86 błędów o poziomie niskim i 51 błędów o poziomie zagrożenia określonym jako informacyjny. Jako ciekawostkę można dodać, iż w dziesięciu kluczowych witrynach internetowych przygotowanych na UEFA EURO 2012 zespół CERT znalazł ogółem 108 błędów, z czego 13 błędów o poziomie zagrożenia bardzo wysokim i 41 błędów o poziomie wysokim.

Coraz bardziej poszerzająca się kolekcja ataków na aplikacje WWW (*Cross Site Scripting*, *Cross Site Request Forgery*, *Directory Traversal*, *XPATH Injection*, *JavaScript Injection*, *Remote File Inclusion*, *SQL Injection*) może być również argumentem świadczącym o ciągłym ewoluowaniu problemu.

Znaczenie wymienionych wcześniej atrybutów bezpieczeństwa oraz sposobów ich zachowania w odniesieniu do systemów informacyjnych opisywane jest w różnych dokumentach, np. w brytyjskich normach z serii BS 7000 [14] i w *Common Criteria* (ISO/IEC 15408) [5], [2]. Od pewnego czasu można również zauważyć tendencję do przekładania ich treści na normy polskie (PN-ISO/IEC 11799:2007 [7], PN-ISO/IEC 27001:2007 [8]). Jednak tendencja ta dotyczy głównie wysokopoziomowych (pod względem szczegółowości i złożoności) norm opisujących przede wszystkim wytyczne dotyczące pożądanego stanu, bez zagłębiania się w szczegóły techniczne problemu w szczególności nie podaje się wskazówek, jak sprawdzać, czy stan ten został osiągnięty. Jest to zrozumiałe, ponieważ zadaniem norm i standardów jest przedstawienie problematyki na poziomie ogólnym (tzn. obejmującym swoim zakresem różne systemy informatyczne) w taki sposób, aby była ona zrozumiała również dla osób niezwiązanych ściśle z technologią, zwłaszcza informatyczną. Stąd w normach powszechne są braki ustandaryzowanych działań dla konkretnych obszarów bezpieczeństwa, które wskazałyby personelowi technicznemu sposób rozwiązywania danych problemów oraz sprawdzania powodzenia przeprowadzonych działań.

W szczególności sytuacja ta dotyczy opisów testów bezpieczeństwa, bowiem prawie nie ma tych opisów w języku polskim. W związku z powyższym celowe wydaje się podjęcie próby opracowania metodyki testów bezpieczeństwa aplikacji internetowych, która będzie zawierała nie tylko wskazówki, co trzeba

zrobić, ale i w jaki sposób przeprowadzić czynności, aby założony cel został osiągnięty.

W Polsce problematykę związaną z metodykami testów poruszają głównie ośrodki akademickie. Dobrym przykładem jest tu metodyka przeprowadzania testów penetracyjnych systemów teleinformatycznych P-PEN [6] opracowana w Wojskowej Akademii Technicznej. Autor metodyki we wstępie podkreśla problem związany z testami bezpieczeństwa: „Testy penetracyjne są postrzegane nawet w środowisku specjalistów jako zajęcie bliższe sztuce, niż rzemiosłu. Dla zleceniodawcy oznacza to niestety całkowity brak kontroli nad tym, co właściwie zrobi wykonawca testów penetracyjnych, a także, co znacznie gorsze, brak informacji zarówno o wnikliwości, jak i skuteczności badań".

Można stwierdzić, że zapotrzebowanie na opracowanie skutecznej metodyki testów bezpieczeństwa aplikacji internetowych (WWW) może pozytywnie wpłynąć na poziom ochrony zasobów informacyjnych. Potrzeba opracowania wspomnianej metodyki wynika z następujących czynników:

- dynamicznego wzrostu popularności sieci Internet oraz powszechności usług WWW w coraz to nowych sferach życia człowieka;
- dynamicznie zmieniających się zagrożeń dla atrybutów bezpieczeństwa informacji udostępnianych z wykorzystaniem serwisów WWW;
- niedoskonałości istniejących metodyk w obszarze testowania bezpieczeństwa aplikacji internetowych (WWW), które nie wskazują na to, w jaki sposób wykonywać takie testy, a przede wszystkim odnoszą się do „głównych punktów" czy wytycznych;
- komercyjnego ukierunkowania metodyk testów, które zazwyczaj są związane z pewnymi firmami lub instytucjami. Takie „firmowe" metodyki są rzadko upubliczniane, co uniemożliwia otwartą dyskusję na temat ich wad i zalet;
- konieczności sformalizowania procesu wykrywania podatności w aplikacjach internetowych;
- braku spójnych metodyk przedstawiających jeden konkretny problem oraz niekompletności i chaosu twórczego w dostępnych zazwyczaj w sieci tzw. listach kontrolnych (ang. *checklists*) itp.;
- braku literatury opisującej konkretnie metodyki testów bezpieczeństwa aplikacji;
- braku tego typu metodyk w języku polskim, co ułatwiłoby również osobom zajmującym się prywatnie, hobbystycznie wykonaniem przeglądu stanu bezpieczeństwa swojego serwisu internetowego.

Należy jednak pamiętać, że podejście metodologiczne musi być traktowane jako zestaw podstawowych i obowiązkowych czynności wykonanych w trakcie testów bezpieczeństwa, który w zależności od obiektu

i szczegółowych celów testów powinien być uzupełniony o niuanse związane właśnie z tymi potrzebami i dodatkową wiedzą ekspercką.

2. Charakterystyka testów bezpieczeństwa aplikacji internetowych (WWW)

W tym punkcie wymienione zostały powody, dla których aplikacje internetowe powinny być poddawane testowaniu. Po uzasadnieniu potrzeby prowadzenia takich działań, autorzy przedstawili różne typy testów bezpieczeństwa aplikacji internetowych, wymienili najczęściej występujące podatności aplikacji internetowych oraz sposoby ich identyfikacji.

2.1. Cele testów bezpieczeństwa aplikacji internetowych

Główne cele, które stawiane są testom bezpieczeństwa aplikacji WWW według pracy [1] są następujące:

- spełnienie regulacji prawnych,
- spełnienie wymagań bezpieczeństwa,
- dopuszczenie aplikacji do użytku.

Poniżej przedstawiono rozwinięcie wymienionych celów testowania zabezpieczeń.

Spełnienie regulacji prawnych

Omawianie wymagań bezpieczeństwa należy rozpocząć od zobowiązań wynikających z regulacji prawnych, aktów normatywnych, ogólnoświatowych standardów, np. PCI DSS[2]. W Polsce założenia i wymagania dotyczące bezpieczeństwa teleinformatycznego regulują m.in. następujące akty prawne:

- Rozporządzenie Prezesa Rady Ministrów z 25.02.1999 w sprawie podstawowych wymagań bezpieczeństwa systemów i sieci teleinformacyjnych;
- Rozporządzenie Ministra Spraw Wewnętrznych i Administracji z dnia czerwca 1998 r. w sprawie określenia podstawowych warunków technicznych i organizacyjnych, jakim powinny odpowiadać urządzenia i systemy informatyczne służące do przetwarzania danych osobowych (znowelizowane Rozporządzeniem Prezesa Rady Ministrów z dnia 1 października 2001 r.);

[2] *Payment Card Industry Data Security Standard* – standard związany z bezpieczeństwem w środowiskach, w których przetwarzane są dane posiadaczy kart płatniczych.

- Rozporządzenie Prezesa Rady Ministrów z dnia 27 kwietnia 2011 r. w sprawie przygotowania i przeprowadzania kontroli stanu zabezpieczenia informacji niejawnych;
- Ustawa z dnia 5 sierpnia 2010 r. o ochronie informacji niejawnych;
- Ustawa z dnia 29 sierpnia 1997 r. o ochronie danych osobowych (znowelizowana: Dz. U. z 2010 r. nr 229, poz. 1497);
- Ustawa o podpisie elektronicznym z 18 września 2001 r. (znowelizowana: Dz. U. z 2001 r. nr 130, poz. 1450, z późn. zm);
- Ustawa o prawie autorskim i prawach pokrewnych z dnia 4 lutego 1994 r.
- Ustawa o świadczeniu usług drogą elektroniczną z dnia 9 września 2002 r. (znowelizowana: Dz. U. 2011.134.779);
- Ustawa z dnia 16 lipca 2004 r. Prawo telekomunikacyjne;
- Ustawa z dnia 12 lutego 2010 r. o zmianie ustawy o informatyzacji działalności podmiotów realizujących zadania publiczne oraz niektórych innych ustaw.

Warto dodać, iż oprócz wymienionych dokumentów podstawowych w praktyce funkcjonuje wiele przepisów i zarządzeń branżowych (bankowych, wojskowych, Agencji Bezpieczeństwa Wewnętrznego) regulujących dokładnie poszczególne zakresy tematyczne bezpieczeństwa teleinformatycznego [4].

Spełnienie wymagań bezpieczeństwa

Słownik do standardu ANSI/IEEE numer 79 (ang. *Standard Glossary of Software Engineering Terminology*) określa [20] wymagania bezpieczeństwa jako:

- warunek lub zdolność pozwalającą użytkownikowi na rozwiązanie problemu lub osiągnięcie jakiegoś celu;
- warunek lub zdolność, które odpowiednio muszą zostać spełnione lub które system bądź jego komponent muszą posiadać w celu spełnienia umowy, standardu, specyfikacji lub innego formalnego dokumentu;
- udokumentowaną reprezentację warunku lub zdolności opisanych w punktach powyżej.

Wspomniane sprawdzenie spełnienia wymagań bezpieczeństwa realizuje się dwojako. Pierwszym sposobem jest dostarczanie dowodów odpowiedniej budowy aplikacji i skuteczności zastosowanych mechanizmów zabezpieczających, np. wykazując odpowiednie metody szyfrowania. Drugim sposobem jest wykazanie nieskuteczności zabezpieczeń i obecności podatności, które można wykorzystać w celu zagrożenia atrybutom bezpieczeństwa. Wykazanie podatności i nieskuteczności zastosowanych zabezpieczeń jest zwykle trudniejsze do przeprowadzenia i wymaga od testujących większej wiedzy. W celu dostarczenia dowodów istnienia oraz możliwości wykorzystania

podatności należy ustalić dane wejściowe, określić spodziewane wyniki i wykonać testy sprawdzające.

Dobrą praktyką jest uwzględnienie wymagań bezpieczeństwa dla tworzonego oprogramowania już w fazie tworzenia koncepcji dla danej aplikacji (*Software Development Life Cycle – SDCL*). W szczególności naturalne powinno być przekonanie o konieczności opracowania takich wymagań na etapie tworzenia projektu.

Należy zauważyć, iż wymaganiami formalnymi oprócz wymagań bezpieczeństwa mogą być również inne standardy techniczne czy dobre praktyki inżynierskie.

Dopuszczenie aplikacji do użytku

Na koniec przybliżony zostanie być może najmniej intuicyjny cel, który został określony jako dopuszczenie aplikacji do użytku. Cel tych testów jest ściśle związany ze stroną biznesową danej aplikacji, czyli oddaniem poprawnie działającej aplikacji internetowej do eksploatacji w środowisku produkcyjnym. Ma to szczególne znaczenie dla aplikacji spełniających funkcje krytyczne dla biznesu. Ryzyko istnienia w aplikacji podatności, której wykorzystanie może przerwać ciągłość działania krytycznej funkcji jest nie do przyjęcia. W praktyce okazuje się, że niektóre aplikacje nie podlegają regulacjom prawnym, a także nie mają stworzonych wymagań bezpieczeństwa. Aplikacja taka może mieć jednak istotne znaczenie biznesowe. Przeprowadzenie takich testów będzie wtedy swego rodzaju zabezpieczeniem przed możliwością kompromitacji aplikacji, która być może w konsekwencji pozwoli na kompromitację innych elementów infrastruktury, bądź nawet na przerwanie jakiegoś procesu biznesowego. Oczywiście testy te powinny uwzględniać wymagania prawne obowiązujące daną aplikację.

Testy bezpieczeństwa powinny być nieodłącznym elementem wszystkich faz SDCL, w szczególności przed udostępnianiem aplikacji do użytku.

2.2. Rodzaje testów bezpieczeństwa aplikacji internetowych

Testy aplikacji internetowych można podzielić na rodzaje, biorąc pod uwagę następujące kryteria:

a) poziom posiadanej wiedzy,
b) zakres testów,
c) lokalizację testów,
d) skład zespołu testowego,
e) sposób przeprowadzania testów,
f) umiejscowienie w cyklu SDCL.

Poniżej przedstawiono klasyfikację rodzajów testów według wyżej wymienionych kryteriów:

a) poziom posiadanej wiedzy:
   - metoda „białej skrzynki" (ang. *„white box" method*) – metoda, w której osoba testująca posiada pełny dostęp zarówno do dokumentacji, jak i kodów źródłowych aplikacji,
   - metoda „szarej skrzynki" (ang. *„gray box" method*) – metoda mieszana, obejmująca wgląd do struktur danych (osoba testująca nie ma dostępu do kodów źródłowych) w fazie przygotowywania testów, sam proces testowania przebiega na zasadzie czarnej skrzynki,
   - metoda „czarnej skrzynki" (ang. *„black box" method*) – metoda, w której oprogramowanie traktowane jest przez osobę testującą jako „czarna skrzynka", o której zawartości nie posiada ona wiedzy, a ze sposobem, w jaki ona działa zapoznaje się w trakcie trwania testów;

b) zakres testów:
   - testy inwazyjne (ang. *invasive*) – testy które polegają na symulacji rzeczywistego ataku. Mogą polegać przykładowo na zablokowaniu testowanej usługi, pozyskaniu lub modyfikacji danych lub innych czynnościach, które w konsekwencji mogą wpłynąć na integralność systemu,
   - testy nieinwazyjne (ang. *non-invasive*) – testy, które są przeciwieństwem testów inwazyjnych. Między innymi mogą polegać na wyszukiwaniu podatności bez praktycznego weryfikowania ryzyka z nimi związanego. Przykładem testów nieinwazyjnych jest analiza kodu źródłowego aplikacji;

c) lokalizacja testów:
   - testy wewnętrzne (ang. *internal*) – testy, które w przypadku aplikacji webowych mogą być testami dostępnych aplikacji, np. w firmowej sieci intranet, z pominięciem dodatkowych zabezpieczeń, takich jak zapory sieciowe, filtry aplikacyjne itp.,
   - testy zewnętrzne (ang. *external*) – testy, które dotyczą wykonywania zdalnych testów poprzez np. sieć Internet oraz zabezpieczenia, które chronią przed działaniami inicjowanymi przez użytkowników sieci Internet;

d) skład zespołu testowego:
   - testy realizowane za pomocą zasobów wewnętrznych firmy, tj. bez konieczności korzystania z zewnętrznych specjalistów,
   - testy przeprowadzane przez firmy zewnętrzne lub wyspecjalizowane zespoły *tiger team* [10] bądź *red team* [9],
   - testy mieszane, w których biorą udział zarówno pracownicy danej firmy, jak i zewnętrzni specjaliści;

e) sposób przeprowadzania testów:
   - testy manualne (ang. *manual*) – testy, które są rozumiane jako testy przeprowadzane ręcznie przez testującego, wymagające specjalistycznej wiedzy. Czasami są one nazywane testami eksperckimi,
   - testy automatyczne (ang. *automatic*) – testy, które mogą odbywać się z użyciem przeznaczonych do tego celu narzędzi automatyzujących pracę testera,
   - testy mieszane (ang. *mixed*) – testy z zastosowaniem obydwu wspomnianych metod;
f) umiejscowienie w cyklu SDCL:
   - testy realizowane w trakcie tworzenia oprogramowania (podejście znane jako *test driven development*),
   - testy wykonywane po zakończeniu fazy tworzenia oprogramowania, czyli po zrealizowaniu czynności programistycznych, bądź w fazie eksploatacji – tradycyjne testy penetracyjne. Innym określeniem tego typu testów będzie atestowanie określane jako „(…) ocena systemu lub komponentu, po zakończeniu procesu jego rozwoju, na zgodność z wyspecyfikowanymi wymaganiami" [27]. Czasami testy te są też nazywane testami akceptacyjnymi.

2.3. Popularne podatności aplikacji internetowych

Wynikiem przeprowadzenia testów bezpieczeństwa aplikacji internetowej jest lista potencjalnych podatności. Należy pamiętać, że podatności związane z aplikacjami WWW zmieniają się, ściśle podążając za rozwojem technologii internetowych.

Do głównych obszarów, związanych z podatnościami aplikacji internetowych należą:

- walidacja danych, np.: niepoprawna walidacja danych wejściowych, „wstrzykiwanie" złośliwego kodu, skrypty krzyżowe, złośliwe wykonywanie plików i binariów, fałszowanie żądań;
- uwierzytelnianie i autoryzacja, np.: niezabezpieczone bezpośrednie odwołania do obiektów, zła obsługa uwierzytelnienia i sesji;
- uprawnienia, np.: niepoprawne uprawnienia do zasobów, brak zabezpieczenia dostępu poprzez URL, niewłaściwe indeksowanie plików i katalogów;
- dostępność usług realizowanych przez aplikacje internetowe, np.: brak odporności na ataki typu DoS i DDoS;

- wycieki informacji, np.: niezabezpieczona wymiana informacji, brak szyfrowania wrażliwych danych lub niepoprawna obsługa błędów, komentarze w kodzie źródłowym, błędna konfiguracja;
- fałszowanie informacji, np.: podszywanie się pod użytkowników aplikacji, *spoofing*, ataki typu *man in the middle*;
- niepoprawna logika, np.: błędy związane z logiką biznesową (co niestety jest często pomijane w tego rodzaju testach).

Kategoryzacji błędów bezpieczeństwa aplikacji internetowych dokonały dwie organizacje: SANS (*SysAdmin, Audit, Network, Security Instituite*) oraz OWASP (*Open Web Application Security Project*).

W tabeli 1 przedstawiono listę 25 najgroźniejszych błędów oprogramowania według instytutu SANS (ang. *2011 CWE /SANS Top 25 Most Dangerous Software Errors*) [18], dane pochodzą z dnia 13.09.2011 r. Celowo w tabelach zostawiono opisy w języku angielskim, ponieważ wiele nazw podatności nie zostało przetłumaczonych w sposób ustandaryzowany na język polski. W związku z tym przetłumaczenie ich przez autorów mogłaby spowodować niejednoznaczności lub wręcz pomyłki w interpretacji tych informacji.

Jako najbardziej niebezpieczne SANS uznał podatności typu *SQL Injection*, którym przypisany został najwyższy wskaźnik istotności wynoszący 93,8.

Organizacja OWASP podzieliła błędy na dziesięć różnych kategorii. Tabela 2 przedstawia kategorie błędów według organizacji OWASP [23].

Tabela 3 przedstawia natomiast porównanie kategorii *OWASP Top Ten 2010* do *CWE/SANS 2010 Top 25* [17].

Analizując porównanie umieszczone w tabeli 3, można zauważyć, iż błędy wymienione przez SANS są bardziej precyzyjnie sformułowane i pokrywają większy obszar (np. błędy przepełnienia bufora nie zostały ujęte w zestawieniu OWASP Top Ten 2010). Dodatkową różnicą są etykiety przyjęte do ich identyfikacji, np. dla błędów określonych przez SANS będzie to CWE (ang. *Common Weakness Enumeration*).

2.4. Sposoby identyfikacji podatności aplikacji WWW

Istnieje wiele różnych sposobów pozwalających na identyfikację oraz rozpoznanie potencjalnych podatności systemów w zakresie bezpieczeństwa, jednak wszystkie służą wymienionym (w pkt. 2.1) celom: spełnieniu wymagań bezpieczeństwa, spełnieniu obowiązujących regulacji prawnych, udostępnieniu aplikacji do użytku.

Tab. 1. Ranking 25 najbardziej groźnych błędów oprogramowania według CWE/SANS [13]

Lp.	Istotność	ID	Podatność
1	93.8	*CWE-89*	*Improper Neutralization of Special Elements used in an SQL Command ("SQL Injection")*
2	83.3	*CWE-78*	*Improper Neutralization of Special Elements used in an OS Command ("OS Command Injection")*
3	79.0	*CWE-120*	*Buffer Copy without Checking Size of Input ("Classic Buffer Overflow")*
4	77.7	*CWE-79*	*Improper Neutralization of Input During Web Page Generation ("Cross-site Scripting")*
5	76.9	*CWE-306*	*Missing Authentication for Critical Function*
6	76.8	*CWE-862*	*Missing Authorization*
7	75.0	*CWE-798*	*Use of Hard-coded Credentials*
8	75.0	*CWE-311*	*Missing Encryption of Sensitive Data*
9	74.0	*CWE-434*	*Unrestricted Upload of File with Dangerous Type*
10	73.8	*CWE-807*	*Reliance on Untrusted Inputs in a Security Decision*
11	73.1	*CWE-250*	*Execution with Unnecessary Privileges*
12	70.1	*CWE-352*	*Cross-Site Request Forgery (CSRF)*
13	69.3	*CWE-22*	*Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal")*
14	68.5	*CWE-494*	*Download of Code Without Integrity Check*
15	67.8	*CWE-863*	*Incorrect Authorization*
16	66.0	*CWE-829*	*Inclusion of Functionality from Untrusted Control Sphere*
17	65.5	*CWE-732*	*Incorrect Permission Assignment for Critical Resource*
18	64.6	*CWE-676*	*Use of Potentially Dangerous Function*
19	64.1	*CWE-327*	*Use of a Broken or Risky Cryptographic Algorithm*
20	62.4	*CWE-131*	*Incorrect Calculation of Buffer Size*
21	61.5	*CWE-307*	*Improper Restriction of Excessive Authentication Attempts*
22	61.1	*CWE-601*	*URL Redirection to Untrusted Site ("Open Redirect")*
23	61.0	*CWE-134*	*Uncontrolled Format String*
24	60.3	*CWE-190*	*Integer Overflow or Wraparound*
25	59.9	*CWE-759*	*Use of a One-Way Hash without a Salt*

Tab. 2. Ranking 10 najbardziej groźnych błędów aplikacji WWW według OWASP w roku 2010 [23]

Lp.	ID	Podatność
1	A1	*Injection*
2	A2	*Cross-Site Scripting (XSS)*
3	A3	*Broken Authentication and Session Management*
4	A4	*Insecure Direct Object References*
5	A5	*Cross-Site Request Forgery (CSRF)*
6	A6	*Security Misconfiguration*
7	A7	*Insecure Cryptographic Storage*
8	A8	*Failure to Restrict URL Access*
9	A9	*Insufficient Transport Layer Protection*
10	A10	*Unvalidated Redirects and Forwards*

Tab. 3. Porównanie błędów *OWASP Top Ten 2010* do *CWE/SANS 2010 Top 25* [17]

OWASP Top Ten 2010	*SANS 2011 Top 25*
A1 – *Injection*	*CWE-89, CWE-78*
A2 – *Cross Site Scripting (XSS)*	*CWE-79*
A3 – *Broken Authentication and Session Management*	*CWE-306, CWE-307, CWE-798*
A4 – *Insecure Direct Object References*	*CWE-862, CWE-863, CWE-22, CWE-434, CWE-829*
A5 – *Cross Site Request Forgery (CSRF)*	*CWE-352*
A6 – *Security Misconfiguration*	*CWE-250, CWE-732*
A7 – *Insecure Cryptographic Storage*	*CWE-327, CWE-311, CWE-759*
A8 – *Failure to Restrict URL Access*	*CWE-862, CWE-863*
A9 – *Insufficient Transport Layer Protection*	*CWE-311*
A10 – *Unvalidated Redirects and Forwards*	*CWE-601*
(not in 2010 OWASP Top Ten)	*The following CWE entries are not directly covered by the OWASP Top Ten 2010: CWE-120, CWE-134, CWE-807, CWE-676, CWE-131, CWE-190.*

Do podstawowych sposobów identyfikacji podatności należą:

- obserwacja – analiza działania aplikacji, kodów źródłowych, nagłówków żądań i odpowiedzi serwera przekazywanych i zwracanych parametrów, ukrytych pól, dodatkowo wywoływanych skryptów, zdarzeń, w szczególności do celów manipulacji danymi wejściowymi, parametrami (np. parametrami sesjami);
- dekodowanie danych – odkodowanie danych zapisanych w formatach utrudniających ich analizę, np. zakodowanych zgodnie z *base64*;
- automatyczne skanowanie – identyfikacja oprogramowania i znanych podatności, a następnie weryfikacja znalezionych podatności w trakcie tzw. testów eksperckich. Należy dodać, że wyniki pozyskane z automatycznych skanerów podsuwają nierzadko pomysły dodatkowych eksperckich scenariuszy testowych;
- wyszukiwanie wad projektu – pomijanie typowych kroków i obowiązkowych elementów, przewidywanie (odgadywanie) danych uwierzytelniających, nadużywanie powtarzalności, testowanie sposobu generowania liczb losowych bądź wyszukiwanie unikatowych błędów związanych z realizacją funkcji biznesowych systemu (błędy logiki biznesowej, ang. *logic flows*).

Należy jednak zaznaczyć, że przy testowaniu zachowania atrybutów bezpieczeństwa systemu trudno jest dowieść braku istnienia błędów. Tym samym niemożliwe jest przeprowadzenie wyczerpującego w 100% testowania zabezpieczeń – zarówno teoretycznie, jak i praktycznie [3]. W praktyce w przypadku realizacji testów bezpieczeństwa można jedynie mówić o zwiększaniu uzasadnionego zaufania do testowanej aplikacji.

3. Przegląd dostępnych metodyk prowadzenia testów bezpieczeństwa aplikacji internetowych

Dla testów aplikacji internetowych ze względu na ich specyfikę nie ma dostępnych zbyt wielu formalnych metodyk, a w szczególności takich, które wskazują przykładowe sposoby i narzędzia pozwalające osiągnąć założony cel testowania. Dodatkowo metodyki różnią się umiejscowieniem ich w cyklu życia oprogramowania, poziomem szczegółowości lub tzw. „ścieżką techniczną". Jako metodyki uznawane w środowisku osób testujących aplikacje internetowe można wymienić metodykę OSSTMM (*Open Source Security Testing Methodology*) oraz metodykę wspomnianej wcześniej organizacji OWASP (*Open Web Application Security Project*). Innym przykładem takiego podejścia metodycznego może być próba zaimplementowania założeń związanych z procesem przeprowadzania testów bezpieczeństwa do znanej z inżynierii

oprogramowania, tzw. „zwinnej" (ang. *agile*), metodyki zarządzania projektami *Agile Security Testing for Web Based Applications* [15] oraz jej rozszerzenia znanego pod nazwą *Extended Agile Security Testing* (w skrócie EAST). W dalszej części artykułu dokładniej opisano wspomniane metodyki.

3.1. Metodyka OSSTMM

OSSTM jest oficjalną metodyką ISECOM (*Institute for Security and Open Methodologies*) z zakresu wykonywania testów bezpieczeństwa, udostępnioną na licencji Creative Commons 3.0 Attribution.

Aktualnym wydaniem metodyki jest wersja 3.02. Wspomniane wydanie wprowadziło wiele zmian w stosunku do poprzednich wersji w kontekście szeroko pojętego zakresu testów. Metodyka proponuje przeprowadzanie testów wszystkimi dostępnymi kanałami operacyjnymi: ludzkim, fizycznym, bezprzewodowym, telekomunikacyjnym oraz sieci danych. Takie podejście sprawia, że możliwe jest uwzględnianie zmian związanych z rozwojem technologii informatycznych i pozwala na objęcie najnowszych technik, tzw. „chmur obliczeniowych" (ang. *cloud computing*), infrastruktur wirtualnych (ang. *virtual infrastructures*), infrastruktur mobilnych (ang. *mobile communication infrastructures*), czy też innych obszarów: zasobów ludzkich (ang. *human resources*), lokalizacji o podwyższonym bezpieczeństwie (ang. *high-security locations*), wiarygodnych środowisk uruchomieniowych (ang. *trusted computing*) oraz procesów logicznych (ang. *logical processes*).

Główne cele określone przez twórców metodyki OSSTM dotyczą dostarczenia audytorowi wytycznych, które pozwolą na zgromadzenie dowodów na stwierdzenie prawdziwości następujących założeń:

- testy przeprowadzono dokładnie,
- testy obejmowały wszystkie niezbędne kanały,
- założenia testów były zgodne z prawem,
- wyniki testów są mierzalne ilościowo,
- wyniki testów są zgodne i powtarzalne,
- wyniki zawierają wyłącznie fakty, pochodzące z przeprowadzonych badań.

Metodyka OSSTMM podzielona jest na pięć sekcji opisujących następujące zagadnienia:

- zbieranie danych (informacji) niezbędnych do przeprowadzania testów,
- badanie poziomu świadomości bezpieczeństwa wśród pracowników,
- badanie poziomu możliwych nadużyć oraz podatności na techniki socjotechniczne,

- badanie sieci teleinformatycznych, w tym testowanie sieci bezprzewodowych,
- weryfikacja poziomu fizycznego bezpieczeństwa oraz procedur bezpieczeństwa.

Metodyka określa zakres odnoszący się do możliwych sposobów interakcji z testowanymi zasobami, w którym wyróżnia się trzy klasy oraz pięć kanałów (tabela 4).

Metodyka zaleca przeprowadzenie prac przy podzieleniu ich na cztery fazy, w skład których wchodzi łącznie siedemnaście następujących modułów:

A. Faza wprowadzenia (ang. *induction phase*) zawiera moduły:

A.1 – przegląd założeń (ang. *posture review*),

A.2 – przygotowania logistyczne (ang. *logistics*),

A.3 – określanie zakresu weryfikacji (ang. *active detection verification*).

B. Faza interakcji (ang. *interaction phase*):

B.4 – audyt widoczności (ang. *visibility audit*),

B.5 – weryfikacja dostępu (ang. *access verification*),

B.6 – weryfikacja zaufania (ang. *trust verification*),

B.7 – weryfikacja wymogów bezpieczeństwa (ang. *control verification*).

C. Faza badań (ang. *inquest phase*):

C.8 – weryfikacja procesów (ang. *process verification*),

C.9 – weryfikacja konfiguracji / weryfikacja przeszkolenia (ang. *configuration verification / training verification*),

C.10 – weryfikacja własności (ang. *property validation*),

C.11 – przegląd segregacji (ang. *segregation review*),

C.12 – weryfikacja ekspozycji (ang. *exposure verification*),

C.13 – wywiad konkurencyjny (ang. *competitive intelligence scouting*).

D. Faza manipulacji (ang. *intervention phase*):

D.14 – weryfikacja kwarantanny (ang. *quarantine verification*),

D.15 – audyt uprawnień (ang. *privileges audit*),

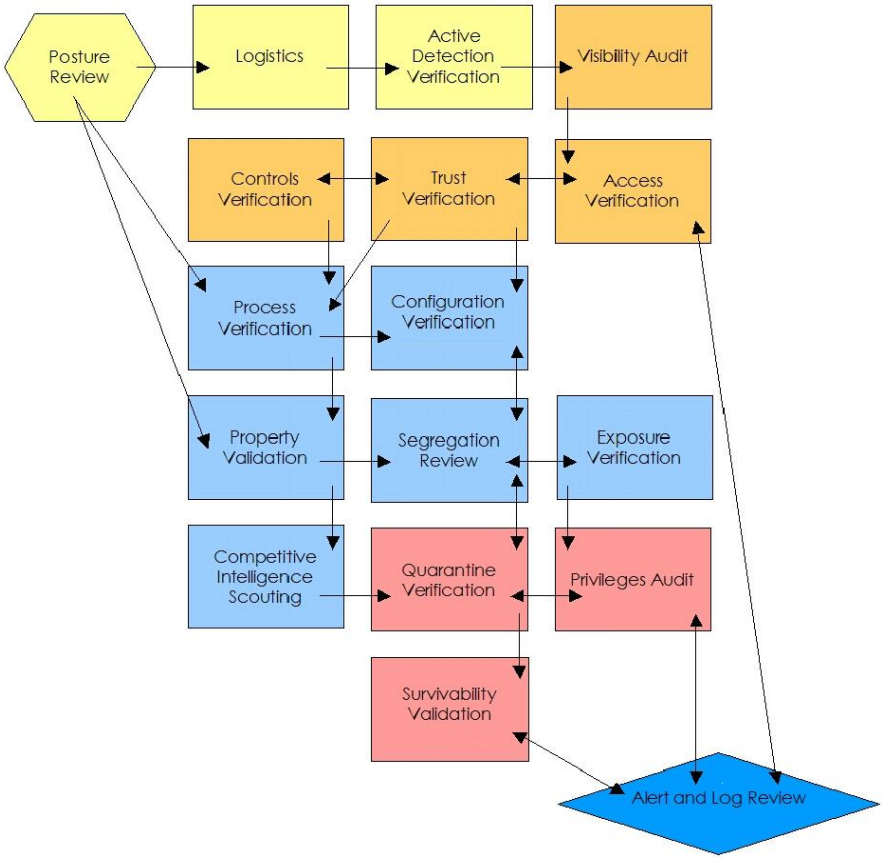
D.16 – potwierdzenie możliwości przetrwania / ciągłość usługowa (ang. *survivability validation / service continuity*),

D.17 – przegląd alarmów i logów / końcowy przegląd (ang. *alert and log review / end survey*).

Kanały i kierunki przepływu informacji pomiędzy poszczególnymi modułami w fazach metodyki OSSTM przedstawia rysunek 1.

Tab. 4. Podział na klasy i kanały wg metodyki OSSTMM [19]

Klasa	Kanał
PHYSSEC (Bezpieczeństwo fizyczne, ang. *physical security*)	Ludzki (ang. *human*)
	Fizyczny (ang. *physical*)
SPECSEC (Bezpieczeństwo widma radiowego, ang. *spectrum security*)	Bezprzewodowy (ang. *wireless*)
COMSEC (Bezpieczeństwo komunikacji, ang. *communications security*)	Telekomunikacyjny (ang. *telecommunications*)
	Sieci danych (ang. *data networks*)



Rys. 1. Kanały i kierunki przepływu informacji pomiędzy poszczególnymi modułami w metodyce OSSTM [19]

Metodyka charakteryzuje się również tym, że wprowadza tzw. metryki bezpieczeństwa (ang. *attack surface metrics*), mogące być podstawą do przeprowadzenia analizy ryzyka, oraz wzór raportu zawierającego wyniki z przeprowadzonych testów – STAR (ang. *the Security Testing Audit Report*).

Do automatycznego przeliczania wspomnianych metryk instytut ISECOM udostępnił arkusz podzielony na trzy następujące zakresy:

I. Zakres bezpieczeństwa operacyjnego OPSEC (ang. *operational security*), w skład którego wchodzą:
   - widoczność aktywów (ang. *visibility*),
   - kontrola dostępu (ang. *access*),
   - zaufanie (ang. *trust*).

   W wyniku sumy trzech wymienionych składników obliczana jest tzw. całkowita porowatość (ang. *total porosity*).

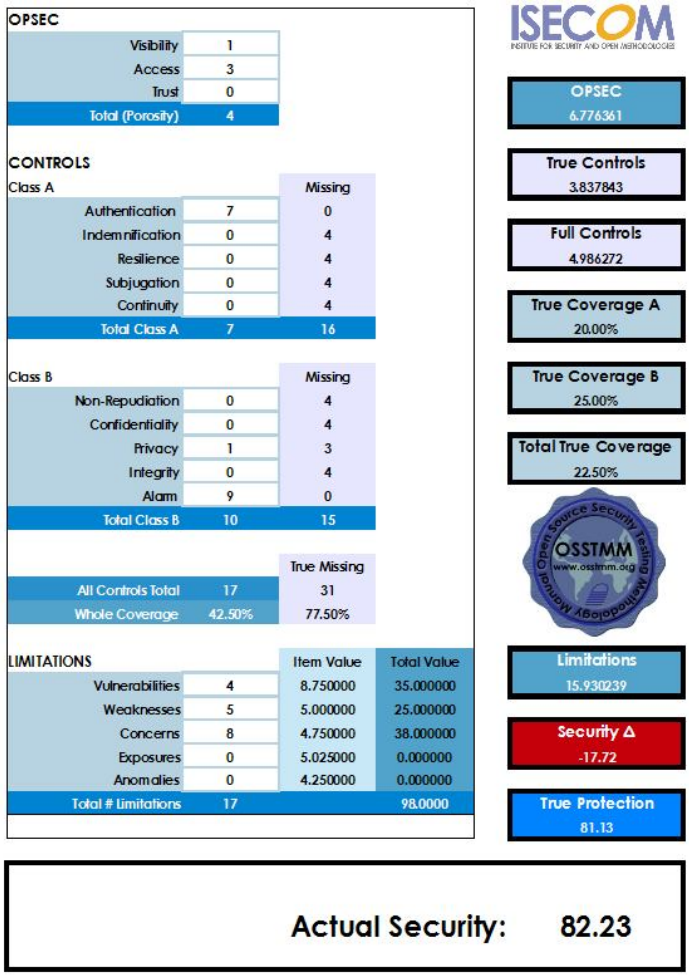
II. Zakres kontroli (ang. *controls*) zawierający dwie następujące klasy:
   - klasa A (kontrole interaktywne): uwierzytelnianie (ang. *authentication*), odszkodowanie (ang. *indemnification*), obciążenie (ang. *resilience*), podporządkowanie (ang. *subjugation*), ciągłość (ang. *continuity*),
   - klasa B (kontrole procesowe): niezaprzeczalność (ang. *non-repudiation*), poufność (ang. *confidentiality*), prywatność (ang. *privacy*), integralność (ang. *integrity*), alarmowanie (ang. *alarm*).

III. Zakres ograniczeń (ang. *limitiations*), który obejmuje:
   - podatności (ang. *vulnerabilities*),
   - słabości (ang. *weaknesses*),
   - obawy (ang. *concerns*),
   - ujawnienia (ang. *exposures*),
   - anomalia (ang. *anomalies*).

Rysunek 2 [19] przedstawia widok arkusza kalkulacyjnego dostępnego na stronie ISECOM, który służy do ustalania udziału elementów porowatości, kontroli i ograniczeń w wypadkowym współczynniku bezpieczeństwa.

Do obliczenia wartości przypisanych do konkretnych kategorii zostały skonstruowane odpowiednie wzory matematyczne. Wadą metodyki są nieliczne wskazania co do narzędzi, które mogą zostać użyte do jej prawidłowej realizacji oraz brak szczegółowych opisów związanych z jej przebiegiem.

Wzór raportu zbudowany jest w taki sposób, aby przedstawić jego odbiorcy dokładną listę zadań, które zostały wykonane w trakcie testów zgodnie z metodyką OSSTMM, z wyraźnym zaznaczeniem tych, których zrealizować się nie udało.



Rys. 2. Widok arkusza kalkulacyjnego dostępnego na stronie ISECOM [19]

3.2. Metodyka OWASP

OWASP jest niekomercyjnym stowarzyszeniem skupiającym grono specjalistów z dziedziny bezpieczeństwa aplikacji internetowych. OWASP w zasadzie nie jest spójną metodyką, lecz zbiorem wytycznych związanych z testowaniem bezpieczeństwa aplikacji. Wytyczne podzielone są na dwie główne kategorie:

- projekty programistyczne (ang. *development projects*),
- projekty dokumentacyjne (ang. *documentation projects*).

Natomiast narzędzia i dokumentacja podzielone są na następujące kategorie:

- *PROTECT* – zbiór narzędzi i dokumentów, które znajdują zastosowanie w zapobieganiu błędom w projektowaniu i implementacji aplikacji,
- *DETECT* – zbiór narzędzi wspomagających znalezienie błędów w projektach i implementacji,
- *LIFE CYCLE* – zbiór narzędzi, który może posłużyć do wzbogacenia cyklu życia rozwoju oprogramowania SDCL o zagadnienia bezpieczeństwa.

Tabela 5 [22] przedstawia wykaz stałych projektów realizowanych w obrębie poszczególnych kategorii. W ramach OWASP występują również projekty o statusach: beta, alfa oraz projekty nieaktywne.

Tab. 5. Kategorie projektów realizowanych przez OWASP

Kategoria	Projekty	Dokumentacja
PROTECT	*OWASP AntiSamy Java Project*	*OWASP Development Guide*
	OWASP AntiSamy .NET Project	*OWASP .NET Project*
	OWASP Enterprise Security API (ESAPI) Project	*OWASP Ruby on Rails Security Guide V2*
	OWASP ModSecurity Core Rule Set Project	*OWASP Secure Coding Practices – Quick Reference Guide*
DETECT	*OWASP JBroFuzz Project*	*OWASP Application Security Verification Standard Project*
	OWASP Live CD Project	*OWASP Code Review Guide*
	OWASP WebScarab Project	*OWASP Testing Guide*
	OWASP Zed Attack Proxy Project	*OWASP Top Ten Project*
LIFECYCLE	*OWASP WebGoat Project*	*OWASP AppSec FAQ Project*
		OWASP Legal Project
		OWASP Source Code Review for OWASP-Projects

W kontekście testowania aplikacji internetowych stosownym podręcznikiem jest *OWASP Testing Guide* [21] z kategorii *DETECT*, koncentrujący się na dostarczeniu procedur testowania bezpieczeństwa aplikacji oraz tzw. listach kontrolnych. Obecne wydanie podręcznika to wersja 3, której edycję zamknięto w 2008 roku (w przygotowaniu jest wersja 4).

Stworzony przez OWASP szkielet procesu testowania (ang. *framework*) obejmuje pięć następujących faz:

1. Przed rozpoczęciem prac programistycznych (ang. *before development begins*).
2. Podczas definiowania i projektowania (ang. *during definition and design*).

3. Podczas prac programistycznych (ang. *during development*).
4. Podczas prac wdrożeniowych (ang. *during deployment*).
5. Podczas utrzymania i eksploatacji (ang. *maintenance and operations*).

Metodyka testów penetracyjnych aplikacji webowych według OWASP jest metodyką bazującą na testach typu *black-box* i obejmującą 66 badań związanych z wymienionymi poniżej dziewięcioma zakresami testów, które poprzedzone są zbieraniem informacji (ang. *information gathering*) na temat obiektu badań. Do wspomnianych wyżej zakresów testów należą:

- testowanie mechanizmów zarządzania konfiguracją (ang. *configuration management testing*),
- testowanie uwierzytelniania (ang. *authentication testing*),
- testowanie zarządzania sesją (ang. *session management testing*),
- testowanie autoryzacji (ang. *authorization testing*),
- testowanie logiki biznesowej (ang. *business logic testing*),
- testowanie walidacji danych (ang. *data validation testing*),
- testowanie DoS (ang. *Denial of Service testing*),
- testowanie usług webservices (ang. *web services testing*),
- testowanie aplikacji AJAX (ang. *AJAX testing*).

Do przykładowych testów kategorii testów zarządzania sesją należą:

- testowanie schematu zarządzania sesją (ang. *testing for session management, schema, ref: OWASP-SM-001*),
- testowanie atrybutów cookies (ang. *testing for cookies attributes, ref: OWASP-SM-001*),
- testowanie możliwości przejęcia sesji (ang. *testing for session fixation, ref: OWASP-SM-003*),
- testowanie eksponowanych zmiennych sesyjnych (ang. *testing for exposed session variables, ref: OWASP-SM-004*),
- testowanie metody ataku CSRF (ang. *testing CSRF, ref: OWASP-SM-005*).

Metodyka testów OWASP wprowadza również sposób szacowania ryzyka (ang. *OWASP risk rating methodology*) oraz przedstawia wytyczne do tworzenia raportu z testów penetracyjnych. W nowej wersji *OWASP Testing Guide* zaplanowane jest między innymi uaktualnienie listy dostępnych podatności.

Istotną zaletą tej metodyki jest to, że opiera się ona na otwartej społeczności specjalistów zainteresowanych zagadnieniami bezpieczeństwa.

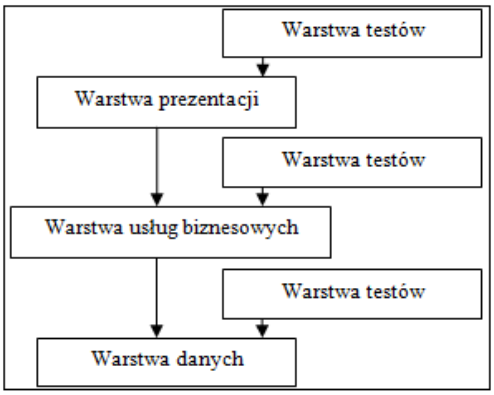
Należy jednak zwrócić uwagę na to, że dostęp do stron internetowych OWASP zawierających zarówno listy kontrolne, jak i przykłady zastosowania oraz narzędzia wspomagające testy posiada dowolny użytkownik sieci Internet (również posiadający złe zamiary).

3.3. Inne podejścia do testowania aplikacji internetowych

Innym podejściem do testów bezpieczeństwa aplikacji internetowych może być metodyka zaliczana do grupy tzw. metodyk „zwinnych" (ang. *Agile Security Testing for Web Based Applications* – w skrócie „agile"). Wywodzi się ona od pojęcia „zwinne programowanie" (ang. *agile software development*), określającego grupę metodyk wytwarzania oprogramowania opartego na programowaniu iteracyjnym. W takim podejściu każdy z członków zespołu projektowego bierze na siebie odpowiedzialność za zadania postawione w każdej iteracji. Ideą *Agile Security Testing for Web Based Applications* jest dostarczenie rozwiązań bezpieczeństwa poprzez transformację wymagań bezpieczeństwa na automatyczne przypadki testowe. Metodyka ta różni się od pozostałych innym umiejscowieniem w cyklu SDCL, zakłada tworzenie przypadków testowych jeszcze przed powstaniem systemu (takie podejście znane jest również pod nazwą TDD – ang. *Test Driven Development*).

Metodyka przedstawiona przez dr. Andrew Tappendena [26] składa się z trzech głównych kroków:

1. Stworzenie wymagań bezpieczeństwa na podstawie zweryfikowanych pod kątem zasadności tzw. „opowieści użytkownika".
2. Stworzenie wysoko „testowalnej" architektury (ang. *a highly testable architecture*, rysunek 3), poprzez dodanie nad typowymi warstwami aplikacji webowych (warstwą prezentacji, warstwą usług biznesowych, warstwą danych) nowej warstwy testów. W ten sposób dla każdej z wymienionych warstw tworzony jest zestaw odrębnych testów bezpieczeństwa. Dzięki takiemu podejściu architektura umożliwia zastosowanie wielu różnych technik testowania bezpieczeństwa wewnątrz wielu poziomów testowych.
3. Uruchomienie automatycznych testów bezpieczeństwa (użycia narzędzi automatyzujących testy).



Rys. 3. Wysoko „testowalna" architektura wg metodyki Agile [26]

Alberto Sillitti w swojej książce [13] opisuje rozszerzenie tej metodyki, nazwane w skrócie EAST (ang. *Extended Agile Security Testing*), polegające na wkomponowaniu jej założeń w metodykę Scrum [24]. Metodyka Scrum została wybrana z powodu wiernego odniesienia się do zwinnych metodyk wytwarzania oprogramowania. W EAST można wyodrębnić następujące kroki:

Krok 1 EAST: opracowanie przypadków niewłaściwego użycia (ang. *misuse cases*), które tworzone są w fazie tzw. *pregame* metodyki Scrum. W trakcie tej fazy wszystkie znane wymagania są spisywane i opracowywana jest lista wymagań dla całego produktu. Zapisywane są one w ramach tzw. Rejestru Produktu [12] (ang. *product backlog*). W ramach oddzielnych spotkań tworzone są kolejne podzbiory listy wymagań zapisane w Rejestrze Sprintu[3] (ang. *sprint backlog*). Zawarte tam wymagania przeznaczone są do realizacji w ramach aktualnego Sprintu (iteracji). Przypadki niewłaściwego użycia opracowywane są podczas dzielenia *sprint backlog* na kilka pozycji *backlog* w celu zdefiniowania dokładnej specyfikacji systemu. Jeśli, co się może zdarzyć, zarys przypadków niewłaściwego użycia zostanie opracowany już na etapie tworzenia *product backlog*, to w kolejnych iteracjach przy tworzeniu podzbiorów wymagań (sprint backlog) przypadki te są tylko uszczegółowiane.

Krok 2 EAST: wykonanie testów jednostkowych polegających na automatycznym przeglądzie kodu (EAST krok 3) w fazie rozwoju, testach penetracyjnych systemu po zbudowaniu części lub kilku komponentów systemu w fazie rozwinięcia (EAST krok 5), testach akceptacyjnych bezpieczeństwa bazujących na wymaganiach bezpieczeństwa (EAST krok 1). W kroku tym wykorzystywana jest wysoko „testowalna" architektura.

Krok 3 EAST: automatyczny przegląd kodu w fazie rozwoju. W metodyce Scrum przegląd ten składa się z następujących aktywności: analiza, projektowanie, rozwój, wdrożenie, testowanie i dokumentowanie. W momencie kiedy kod źródłowy systemu jest gotowy, zostaje poddany testom za pomocą narzędzi do statycznej analizy kodu.

Krok 4 EAST: uzupełnienie (aktualizacja) repozytorium wiedzy. Każda decyzja podjęta w fazie rozwoju lub fazie przeglądu powinna zostać udokumentowana. Dodatkowo, jeśli jakieś podatności zostaną wykryte już w trakcie eksploatacji systemu, należy dodać je do repozytorium wiedzy.

[3] Według pracy [12] Sprint jest to ograniczenie czasowe trwające jeden miesiąc lub krócej, podczas którego wytwarzany jest Przyrost „Ukończonej", używalnej i potencjalnie gotowej do uruchomienia funkcji.

Krok 5 EAST: przeprowadzenie testów penetracyjnych i eliminacja błędów typu *false-positive*[4]. Testy penetracyjne w fazie rozwinięcia przeprowadza się za pomocą automatycznych skanerów podatności, a ich wyniki powinny zostać poddane analizie, podczas której oznacza się błędy typu *false-positive*.

Krok 6 EAST: wydanie końcowej oceny i kwalifikacja. Po fazie przeglądu powinna nastąpić sesja oceny końcowej, na której uczestnicy testów rozważają podatności wykryte podczas fazy rozwoju i testowania bezpieczeństwa.

Jeśli chodzi o kategorie błędów użytych w przykładzie zastosowania metodyki, autorzy wspominają o użyciu wymienionych wcześniej „OWASP Top Ten".

Założeniem EAST było rozszerzenie metodyki *Agile Security Testing* o zidentyfikowane obszary wymagające weryfikacji: objęcie testami całego cyklu SDCL, oznaczanie podatności oraz dostarczenie repozytorium testowego. Przez to, że zwinna metodyka EAST ma swoje odzwierciedlenie w każdej z faz projektowych, wydaje się najbardziej wkomponowana w cykl życia oprogramowania. Niestety, EAST nie dostarcza zbioru narzędzi wspomagających prace testera oraz wspomnianego repozytorium testowego (bądź wskazówek, jak takie repozytorium zaprojektować, aby spełniało założenia metodyki). Dodatkowym utrudnieniem dla testera jest brak określonej specyfikacji, w jaki sposób wykonywać testy bezpieczeństwa. W tym zakresie metodyka wskazuje jedynie odwołania do metodyki OWASP, w konsekwencji dając sugestie, by traktowano ją jako uzupełnienie metodyki OWASP.

4. Podsumowanie

W artykule skupiono się na ogólnym przedstawieniu zagadnień związanych z testami bezpieczeństwa aplikacji internetowych oraz założeń stosowanych w tym zakresie metodyk, bez szczegółowej analizy występujących między nimi różnic.

Analiza dostępnych metodyk pozwoliła na wyodrębnienie obszarów wymagających uwzględnienia przez spójną metodykę testów bezpieczeństwa aplikacji internetowych. Do wspomnianych wyżej obszarów należą:

a) zakres (określa zakres i szczegółowość testów bezpieczeństwa);
b) fazy (ilość etapów w metodyce);

[4] Określenie *false-positive* oznacza omyłkowe zaklasyfikowanie poprawnie działającego elementu jako błędu lub podatności.

c) oznaczenie podatności (określony i uwzględniony podział na błędy typu *false/true positive*, wskazane jest podawanie istotności danego błędu. W zakres tego obszaru wchodzi również weryfikowanie podatności oraz przypisanie podatności do odpowiedniej grupy po przeprowadzeniu weryfikacji, np. w ramach testów eksperckich);
d) metryki (propozycja metryk pozwalających przeprowadzić analizę ryzyka);
e) narzędzia (wskazuje specjalizowane narzędzia wspomagające pracę testera);
f) środowisko testowania (propozycja środowiska programowego do przeprowadzania testów);
g) raportowanie (propozycja zawartości raportu z testów);
h) repozytorium (udostępnione repozytorium przechowujące dane testowe – pozwalające testerowi m.in. na zarządzanie przebiegiem testów oraz cyklem życia wykrytych podatności w przypadku testów cyklicznych).

Wspomniane obszary oraz ich uwzględnienie w poszczególnych metodykach zostało przedstawione w tabeli 6 [1].

Komentarza wymaga podpunkt h) (repozytorium) jako nieuwzględniony w dostępnych metodykach testów, a zdaniem autorów pozwalający na uporządkowanie wyników testowych. Jedynie w treści metodyki EAST wspominano o repozytorium testowym, jednak brakuje wskazań co do jego założeń oraz możliwych sposobów implementacji. W konsekwencji uznano, że taka informacja jest mało użyteczna i w związku z tym trudno mówić o uwzględnieniu przez tę metodykę zakresu, jakim jest repozytorium.

Dodatkowo odpowiednio przygotowane repozytorium umożliwia współpracę wielu testerów oraz zapewnia możliwość późniejszego zarządzania informacjami o podatnościach w przypadku powtórnych testów. Repozytorium wprowadza również ułatwienia związane z nadzorowaniem procesu wdrażania poprawek oraz oceną skuteczności podejmowanych działań.

Warto zauważyć, że jedyną metodyką testowania bezpieczeństwa aplikacji WWW mającą swoje korzenie w środowisku akademickim jest Agile Security Testing for Web Based Applications. Zauważalny jest również brak wsparcia normatywnego w tej dziedzinie (w Polsce), które mogłoby stanowić wytyczne dla instytucji/specjalistów zainteresowanych testowaniem bezpieczeństwa aplikacji internetowych. Pozwoliłoby to przede wszystkim na zapewnienie spójności stosowanej w naszym kraju terminologii oraz zbliżenie do siebie różnych metodologii przeprowadzania tego typu testów.

Autorzy są zdania, że testy bezpieczeństwa nie zapewniają kompletnej obrony przed błędami bezpieczeństwa aplikacji internetowych. Są one jednak ważnym składnikiem kompleksowej ochrony, który powinien być traktowany na równi z zabezpieczeniami technicznymi, takimi jak np.: systemy klasy WAF (ang. *Web Application Firewall*), IDS/IPS (ang. *Intrusion Detection System,*

Intrusion Prevention System), NGF (ang. *Next Generation Firewall*). Należy dodać, że testy powinny być wykonywane cyklicznie, odpowiednio do zmian w chronionej aplikacji oraz otaczających ją zagrożeń. Z powyższych powodów istotna wydaje się kwestia wykonywania tych testów w sposób metodyczny, co z kolei implikuje potrzebę opracowania skutecznej, kompletnej i spójnej metodyki testowania aplikacji internetowych.

Tab. 6. Ogólne porównanie dostępnych metodyk [1]

	OSSTM	OWASP	Agile AST	EAST
Zakres	WWW, informacji, sieci, bezp. fizyczne	WWW, kod źródłowy	WWW	WWW
Liczba faz	4	5	3	6
Oznaczanie podatności	brak	brak	brak	uwzględnia
Metryki	*rav*	*Application Security Metrics Project*	brak	brak
Narzędzia	brak[5]	uwzględnia	brak	brak
Środowisko testowe	brak	*O2 Platform Project*	brak	brak
Raportowanie	*STAR (Security Test Audit Report)*	*Uniform Reporting Guidelines*	brak	brak
Repozytorium	brak	brak	brak	brak[6]

Literatura

[1] ANTCZAK M., *Metodyka oraz dedykowany system służący do wykonywania testów bezpieczeństwa aplikacji internetowych (WWW),* praca magisterska, Wojskowa Akademia Techniczna, 2012.

[2] BIAŁAS A., *Bezpieczeństwo informacji i usług w nowoczesnej instytucji i firmie*, Wydawnictwa Naukowo-Techniczne, Warszawa, 2007.

[3] HOPE P., WALTER B., *Testowanie bezpieczeństwa aplikacji internetowych. Receptury*, Helion, Gliwice, 2010.

[4] LIDERMAN K., *Podręcznik administratora bezpieczeństwa teleinformatycznego*, Mikom, Warszawa, 2003.

[5] W kontekście narzędzi do testowania bezpieczeństwa aplikacji internetowych (WWW).

[6] W kontekście dostarczenia gotowego repozytorium bądź wskazania założeń do jego zaprojektowania.

[5] Norma ISO 15408, Common Criteria.

[6] PATKOWSKI A. E., *Metodyka P-PEN przeprowadzania testów penetracyjnych systemów teleinformatycznych*, „Biuletyn Instytutu Automatyki i Robotyki" nr 24/2007, Wojskowa Akademia Techniczna, 2007.

[7] Polska Norma PN-ISO/IEC 11799:2007.

[8] Polska Norma PN-ISO/IEC 27001:2007.

[9] Portal Wikipedia [online]. *Red team – Wikipedia, the free encyclopedia*, 2012, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://en.wikipedia.org/wiki/Red_team

[10] Portal Wikipedia [online]. *Tiger team – Wikipedia, the free encyclopedia*, 2012, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://en.wikipedia.org/wiki/Tiger_team

[11] Portal Wikipedia [online]. „*Webmail – Wikipedia, wolna encyklopedia*", 2012, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://pl.wikipedia.org/wiki/Webmail

[12] SCHWABER K., SUTHERLAND J., *The Scrum Guide*, wersja polska. [dostęp: 19-10-2012]. Dostępna w Internecie: http://www.scrum.org/Scrum-Guides,

[13] SILLITTI A., WANG X., MARTIN A., WHITWORTH E., *Agile Processes in Software Engineering and Extreme Programming*, 11th International Conference, XP 2010, Trondheim, Norway, June 1-4, 2010, Proceedings, pp. 14-26, Springer, 2010.

[14] Standard BS 7000-1:2008, Design management systems. Guide to managing innovation, wyd. BSI, Wielka Brytania, 2008.

[15] Strona Agile Methodology [online]. *Development Methodologies | What Is Agile Methodology*, 2008, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://agilemethodology.org/

[16] Strona CERT.GOV.PL [online]. *Raport z działalności zespołu CERT.GOV.PL za II kwartał 2012 roku*, 2012, [dostęp: 08-10-2012]. Dostępny w Internecie: http://www.cert.gov.pl/portal/cer/56/561/Raport_z_dzialalnosci_zespolu_CERTGOVPL_za_II_kwartal_2012.html

[17] Strona Common Weakness Enumeration [online]. *2011 CWE/SANS Top 25 Most Dangerous Software Errors. Appendix D: Comparison to OWASP Top Ten 2010*, 2012, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://cwe.mitre.org/top25/#AppendixD

[18] Strona Common Weakness Enumeration [online]. *2011 CWE/SANS Top 25 Most Dangerous Software Errors* – wersja z 13 września 2011 roku, [dostęp: 08-10-2012]. Dostępny w sieci Internet: http://cwe.mitre.org/top25/

[19] Strona ISECOM [online], Pete Herzog, *OSSTMM 3. The Open Source Security Testing Methodology manual*, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://www.isecom.org/mirror/OSSTMM.3.pdf

[20] Strona Norwegian University of Science and Technology [online]. *Computer and Information Science, Department of Computer and Information Science*, 2002, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://www.idi.ntnu.no/grupper/su/publ/ese/ieee-se-glossary-610.12-1990.pdf, p. 62.

[21] Strona OWASP Foundation © 2011 [online]. *Category: OWASP Testing Project*, 2011, [dostęp: 09-10-2012]. Dostępny w sieci Internet: https://www.owasp.org/index.php/OWASP_Testing_Project

[22] Strona OWASP Foundation © 2011 [online]. *Category: OWASP Project- OWASP*, 2011, [dostęp: 09-10-2012]. Dostępny w sieci Internet: https://www.owasp.org/index.php/Category:OWASP_Project

[23] Strona OWASP Foundation © 2011 [online]. *Category: OWASP Top Ten Project*, 2011, [dostęp: 09-10-2012]. Dostępny w sieci Internet: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[24] Strona Scrum Methodology [online]. *Scrum Phases overview*, 2009, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://www.scrummethodology.org/scrum-phases.html

[25] Strona The Web Application Security Consortium / Threat Classification [online]. *WASC Threat Classification online*, 2011, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://projects.webappsec.org/w/page/13246978/Threat%20Classification

[26] Strona University of Calgary [online], *Electrical Engineering*. A. Tappenden, P. Beatty, J. Miller , A. Geras, M. Smith, *Agile Security Testing of Web-Based Systems via HTTPUnit*, [dostęp: 09-10-2012]. Dostępny w sieci Internet: http://enel.ucalgary.ca/People/Smith/2008webs/encm515_08/08ReferenceMaterial/Agile%20Security%20Testing%20Miller%20Smith.pdf

[27] SUBIETA K., *Wprowadzenie do inżynierii oprogramowania*, PJWSTK, Warszawa, 2002.

Testing of internet applications security

ABSTRACT: The paper provides an introduction to the issue of web application security testing. Main goals for security tests and known types of tests divided into categories are presented. Areas related to web application vulnerabilities and general methods to identify web application security errors are indicated. A supplementation of nomenclature associated with web application vulnerabilities is presented. A general overview of available security testing methodologies for web applications is given.

KEYWORDS: internet applications security, security testing, security vulnerabilities, methodology of security testing, penetration tests, security attributes

Praca wpłynęła do redakcji: 24.10.2012