

***Testowanie bezpieczeństwa aplikacji
w praktyce***



SecuRing

Wojciech Dworakowski



Agenda

Bezpieczeństwo aplikacji webowych – specyfika problemu

Metody testowania aplikacji

- Testy dynamiczne i statyczne, narzędzia

Typowe podatności i ich testowanie

- Czy automaty dadzą radę?

Co trzeba przemyśleć zanim rozpocznie się testowanie?

Próby standaryzacji

- OWASP ASVS
- OWASP Testing Guide



Trochę statystyki

Większość aplikacji tworzonych na zamówienie posiada istotne podatności (o znacznym wpływie na ryzyko)

Dotyczy to zwłaszcza aplikacji webowych

- Pięta achillesowa współczesnych systemów IT

Threat rank	N of Vulns	N of Sites	N of Sites	% Sites
Urgent	8918	2287	9.14%	18.77%
Critical	44669	5511	45.79%	45.22%
High	35375	8807	36.26%	72.27%
Medium	4908	4455	5.03%	36.56%
Low	3663	3618	3.75%	29.69%

24678 aplikacji przebadanych metodami testu penetracyjnego black-box, white-box oraz skanerami automatycznymi w roku 2008 (8 różnych firm)

Źródło: WASC Web Application Security Statistics

<http://projects.webappsec.org/Web-Application-Security-Statistics>



Z naszych doświadczeń

Testy najczęściej są zamawiane tuż przed wdrożeniem produkcyjnym

- W większości przypadków tylko testy penetracyjne
- Często jest to jedyna forma weryfikacji bezpieczeństwa

Kluczowe podatności znajdujemy w ok. **70% badanych aplikacji**

***Metody testowania
bezpieczeństwa aplikacji***





Metody testowania

o://www.dfa.

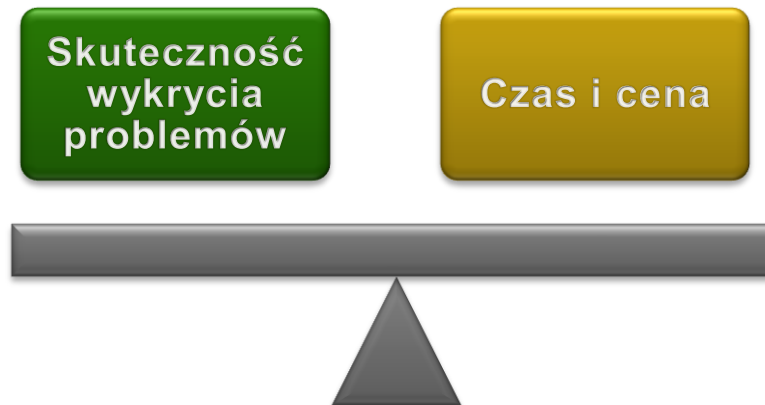
Testy dynamiczne

- testy skanerem zautomatyzowanym
- testy penetracyjne

Testy statyczne

- wywiad
- testy skanerem zautomatyzowanym
- przegląd kodu

Uwaga! Ważny jest efekt końcowy a nie zastosowane metody





Testy penetracyjne

o://www.dfa.

Kontrolowane próby przełamania zabezpieczeń

Symulacja ataku na aplikacje

- Testerzy powinni używać podobnych metod i narzędzi jak potencjalny intruz

Testy blackbox

- identyczny zasób informacji jak potencjalny użytkownik

Testy whitebox

- dostęp do dokumentacji projektowej, kodu źródłowego, konfiguracji, możliwość przeprowadzenia wywiadów

... i odcienie pośrednie



Testy penetracyjne

o://www.dfa.

Zalety

- spojrzenie „z punktu widzenia intruza”
- jak wiele może działać potencjalny intruz? (również: czego może się dowiedzieć o budowie systemu)
- metoda (stosunkowo) szybka
- metoda (stosunkowo) tania

Wady

- nie dają gwarancji wykrycia wszystkich podatności
- zakres ciężki do ścisłego zdefiniowania
- istnieje ryzyko spowodowania awarii
- bardzo dużo zależy od doświadczenia zespołu testowego
- ciężko jest rozliczyć testerów z niewłaściwego wykonania



Skannery zautomatyzowane

o://www.dfa.

Istnieją ...ale:

- Ciężko jest zaprogramować skaner który „przejdzie” całą aplikację
- Specyficzne podatności ciężko jest testować automatem (trudności z zaprogramowaniem warunku powodzenia)
- Nie są (i nie będą) w stanie testować podatności w logice aplikacji
- Cena rzadko kiedy uzasadnia nabycie przez użytkownika końcowego
- Wynik trzeba interpretować (problem false positives)
- Zaleta: test jest powtarzalny



Przegląd kodu źródłowego

o://www.dfa.

Współczesne aplikacje z reguły są dość rozbudowane

W praktyce ciężko jest „z doskoku” zrozumieć duży skomplikowany kod

Możliwe działania wybiórcze

- przegląd funkcji bezpieczeństwa (np. uwierzytelnienie, autoryzacja transakcji)
- przegląd implementacji zabezpieczeń (np. walidacja, szyfrowanie, etc)
- wywiad z developerami wspomagający przegląd kodu



Skannery kodu źródłowego

Zalety:

- są w stanie szybko rozpoznać typowe podatności (typowo zakodowane)
- powtarzalność

Wady:

- Jedną podatność można zakodować na wiele sposobów
- Nie wykryją błędów logicznych
- Nie wykryją błędnej architektury
- Nie wykryją błędów implementacji zabezpieczeń (np. szyfrowania)

Większość produktów raczej przeznaczona dla developera niż dla „audytora”

***Typowe podatności
... i problemy z testowaniem***





OWASP Top 10 (2010 RC1)

A1 - Injection Flaws

A2 - Cross Site Scripting (XSS)

A3 - Broken Authentication and Session Management

A4 - Insecure Direct Object Reference

A5 - Cross Site Request Forgery (CSRF)

A6 – Security Misconfiguration

A7 - Failure to Restrict URL Access

A8 – Unvalidated Redirects and Forwards

A9 - Insecure Cryptographic Storage

A9 – Insufficient Transport Layer Protection



SQL injection

o://www.dfa.

Atak wykorzystujący nieudolne połączenie aplikacji i bazy danych

```
"SELECT * FROM users WHERE name = " +  
request.getParameter("username");
```

Skutki:

- wyciągnięcie dowolnych danych z bazy zasilającej aplikację
- obejście logiki biznesowej (i większości zabezpieczeń)
- ograniczenia: ...doświadczenie i wyobraźnia intruza ;)



SQL injection - testowanie

Trzeba sprawdzić większość parametrów w całości aplikacji (te które są częścią zapytań do bazy)

Trzeba rozpoznać sposób przetwarzania danych przez aplikację

Automat

- wykryje tylko bardzo oczywiste przypadki
- mało który jest w stanie wykryć podatność o ile aplikacja nie reaguje błędem na niepoprawną składnię SQL
- w większości przypadków nie jest w stanie dobrać składni SQL tak żeby zmienione zapytanie było poprawne składniowo



SQL injection - testowanie

o://www.dfa.

Przegląd kodu

- tak – jeśli dostęp do danych jest zorganizowany w sposób jednolity
- można poszukiwać ciągów znaków charakterystycznych dla sklejanego SQL
- da się w miarę dobrze automatyzować (skryptami)

Uwaga na przypadki specjalne !

- np. dynamiczny blok PL/SQL w Oracle
- fantazja developerów nie zna granic ;)



Cross site scripting (XSS)

Aplikacja przyjmuje z przeglądarki wartość parametru i używa tej wartości w treści wygenerowanej strony

- Przykład: krok 1: wpisanie danych przelewu, krok 2: zatwierdzenie przelewu

Podatność = brak walidacji + brak kodowania znaków specjalnych

Skutek: Wywołanie wrogiego kodu JavaScript w przeglądarce ofiary (w kontekście aplikacji)

Reflected XSS

- Użytkownik musi „użyć” wywołania HTTP przygotowanego przez intruza
- W parametrach – wrogie kod (przykład powyżej)
- Kod wykonuje się natychmiast po użyciu

Stored XSS

- Wrogie kod zapisany w danych przechowywanych w aplikacji
- Przykład: XSS w opisie przelewu
- Kod wykona się po obejrzeniu „zarażonych” danych



Reflected XSS – testy blackbox

Trzeba sprawdzić wszystkie parametry w całości aplikacji

Ręcznie – żmudne (ale konieczne)

Automat – wykryje tylko najprostsze przypadki

- jeśli jeden parametr jest nieprawidłowy to najczęściej aplikacja nie przetworzy pozostałych parametrów (walidator)
- automat nie jest w stanie zrozumieć kontekstu kodu w jakim jest używana potencjalnie podatna wartość parametru

Skrypty dostosowane do specyfiki aplikacji



Stored XSS – testy blackbox

Trzeba przeanalizować logikę biznesową aplikacji i zastanowić się które z parametrów mogą:

- Pojawiać się jako dane na stronach innych użytkowników aplikacji
- Pojawiać się w innych miejscach u tego samego użytkownika (mniejsze znaczenie)

Automat – raczej sobie nie poradzi



Cross site scripting (XSS)

- przegląd kodu



Testowanie „case by case”

- dla każdego danych wchodzących do aplikacji
- trudne (trzeba śledzić przepływ danych w aplikacji)

Można sprawdzić czy są stosowane zasad dobrej praktyki

- walidacja
- normalizacja
- kodowanie znaków specjalnych

Da się sprawdzić:

- o ile zastosowano spójne, wymuszane „centralnie” zasady walidacji
- o ile zastosowano jednolitą metodę wypisywania danych na generowanych stronach
- niestety – jest to rzadko spotykane „w naturze”



Nieprawidłowa kontrola dostępu do danych

<http://moj.bank/accountHistory?accountID=1234>

zmienienie accountID na „cudzy” identyfikator

Skutek: podgląd lub modyfikacja cudzych danych



Kontrola dostępu do danych - testowanie

o://www.dfa.

Testy blackbox - Niby oczywiste, ale:

Trzeba odnaleźć wszystkie identyfikatory „globalne”

- ilość (→ czas → pieniądze) zależy od sposobu implementacji aplikacji

Dla każdego z nich spróbować „cudzej” wartości

Automat:

- jak rozróżnić dane „moje” od „cudzych”?

Przegląd kodu:

- sprawdzenie sposobu implementacji kontroli dostępu
- rzadko („nigdy” ?) jest stosowane zunifikowane podejście



Błędy logiczne

Przykład: Autoryzacja transakcji podpisem elektronicznym (kluczem)

- Wydanie nowego klucza nie wymaga autoryzacji starym kluczem

Przykład: Autoryzacja kodem SMS

- Opis transakcji jest ustalany po stronie przeglądarki

Przykład: Zlecenia stałe

- Ustawienie zlecenia co 0 dni → DoS

Automaty nie wykryją tego typu podatności

- Przynajmniej na obecnym poziomie zaawansowania sztucznej inteligencji

***Co trzeba przemyśleć zanim
rozpoczniemy testowanie?***





Zakres testów

Jednoznaczne określenie obiektu testów

- Adres aplikacji
- Wszystkie punkty wejścia do aplikacji

Badania

- Testy penetracyjne
 - Jaki zbiór uprawnień początkowych?
- Przegląd konfiguracji
 - Na zgodność z czym?
- Przegląd kodu źródłowego
 - W jakim zakresie?

Metodyka testowania



Jaki zbiór uprawnień początkowych?

To zależy....

od zagrożeń dla testowanej aplikacji

Przykład: Aplikacja dla doradców klienta

- Anonimowy użytkownik
- Agent indywidualny, Koordynator, Pracownik oddziału franczyzowego, Kierownik,
- Administrator

Warto wykonać analizę zagrożeń

- Jakie grupy użytkowników (role)?
- Jakie zaufanie mamy do tych użytkowników?
- Czy istnieją jakieś zewnętrzne ograniczenia?
- Jakie mogą być skutki nawet jeśli podatności istnieją?



Kiedy testować?

Jeśli tylko testy bezpieczeństwa w ramach testów odbiorczych:

Po testach funkcjonalnych i gdy wprowadzono wszystkie poprawki funkcjonalne

- Każda zmiana może skutkować wprowadzeniem podatności
- Zasada „złotego środka”

Trzeba przewidzieć czas na poprawienie podatności i ponowne ich przetestowanie

- Uwaga: ciężko przewidzieć bo ilość i „ciężar” poprawek jest niewiadoma

Jeśli to możliwe to sprawdzanie bezpieczeństwa rozpocząć wcześniej

- Zdefiniowanie i sprawdzenie założeń dotyczących bezpieczeństwa
- Ocena projektu
- Testy kontrolne w trakcie wykonywania aplikacji



Testowanie na systemie produkcyjnym

Jeśli jest to możliwe to testy bezpieczeństwa powinny być wykonane na systemie testowym

- Osobnym problemem jest zapewnienie spójności wersji

Potencjalne problemy przy testowaniu na systemie produkcyjnym

- Ryzyko destabilizacji systemu
- Ryzyko utraty integralności danych
- Nie da się przetestować wszystkich funkcjonalności (np. kredyty, lokaty, inwestycje, aplikacje maklerskie, etc.)
- Problemy logistyczne (np. odblokowanie konta, przekonfigurowanie praw dostępu)
- Dla niektórych aplikacji nie może być kont testowych na systemach produkcyjnych (np. bankowość internetowa)

Próby standaryzacji





Problem

Rynek usług testowania bezpieczeństwa jest bardzo szeroki

Klient z reguły nie dysponuje wiedzą specjalistyczną

Jak porównać oferty?

- Testowanie prostym narzędziem automatycznym
- Testy „ręczne” + wyczerpujący przegląd kodu

Dla klienta liczy się cena i uzyskana wartość

- Rezultatem testów jest raport ze spisem podatności
- Nie sposób porównać proponowane metodyki
- Jeśli testy będą zbyt ogólne to skąd będzie wiadomo czy wszystkie podatności zostały wykryte?



OWASP Application Security Verification Standard (ASVS)

Cel: Znormalizowanie dostępnych na rynku usług weryfikacji bezpieczeństwa aplikacji

Zajmuje się testowaniem zabezpieczeń, które chronią przed typowymi zagrożeniami

- Celem oceny wg ASVS nie jest poszukiwanie podatności
- ale sprawdzenie czy istnieją odpowiednie zabezpieczenia – zasady dobrej praktyki
- Daje możliwość wypowiedzenia się również o tym „co jest poprawne” – „spojrzenie pozytywne”

Można stosować:

- Jako wzorzec – przy weryfikacji
- Jako wytyczne – dla developerów
- Jako specyfikacja – w kontraktach na wykonanie aplikacji



OWASP ASVS – Poziomy weryfikacji

Poziom 1 – Weryfikacja automatyczna

- 1A – Dynamic Scan
- 1B – Source Code Scan

Poziom 2 – Weryfikacja ręczna

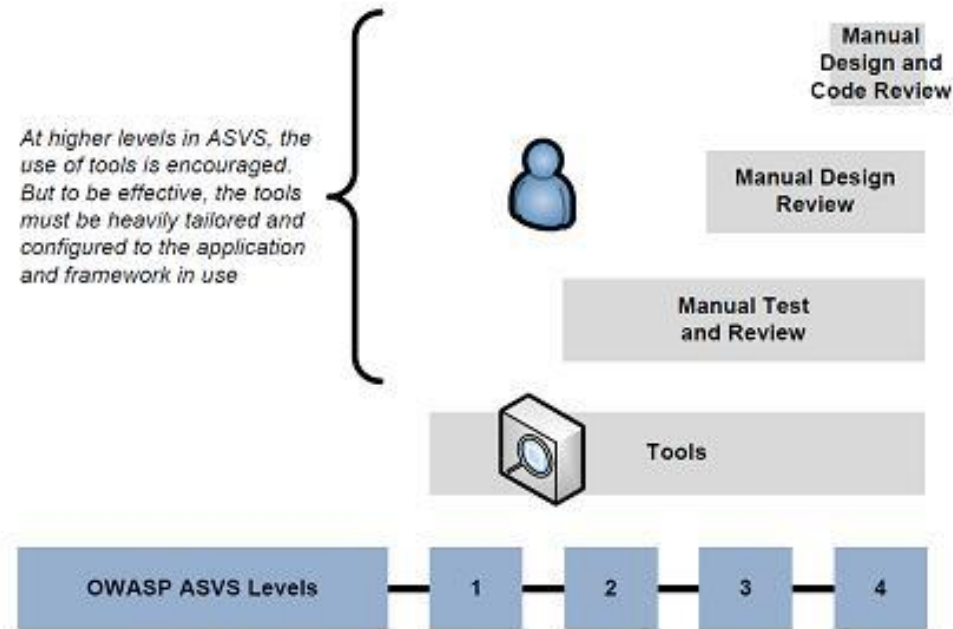
- 2A – Penetration Test
- 2B – Code Review

Poziom 3 – Weryfikacja projektu

- L2 + wszystkie biblioteki i serwisy + modelowanie zagrożeń i weryfikacja projektu

Poziom 4 – Weryfikacja wewnętrzna

- L3 + framework, narzędzia, etc + weryfikacja możliwości wprowadzenia złośliwego kodu





OWASP ASVS - Raport

o://www.dfa.

ASVS definiuje co ma być zawarte w raporcie

- Wprowadzenie
- Opis aplikacji
- Opis architektury
- Rezultat sprawdzeń (lista kontrolna)



Sprawdzenia - przykład

Table 2 - OWASP ASVS Authentication Requirements (V2)

Verification Requirement	Level 1A	Level 1B	Level 2A	Level 2B	Level 3	Level 4
V2.1 Verify that all pages and resources require authentication except those specifically intended to be public.	✓	✓	✓	✓	✓	✓
V2.2 Verify that all password fields do not echo the user's password when it is entered, and that password fields (or the forms that contain them) have autocomplete disabled.	✓	✓	✓	✓	✓	✓
V2.3 Verify that if a maximum number of authentication attempts is exceeded, the account is locked for a period of time long enough to deter brute force attacks.	✓		✓	✓	✓	✓
V2.4 Verify that all authentication controls are enforced on the server side.			✓	✓	✓	✓
V2.5 Verify that all authentication controls (including libraries that call external authentication services) have a centralized implementation.				✓	✓	✓
V2.6 Verify that all authentication controls fail securely.			✓	✓	✓	✓
V2.7 Verify that the strength of any						



OWASP ASVS - Uwagi

Dostępność kompatybilnych narzędzi

- Na poziomie 1A i 1B
- Wg badań MITRE narzędzia automatyczne wykrywają (w sumie) tylko 45% znanych typów podatności
 - przy czym część wspólna jest niewielka
- Pojawiają się narzędzia „ASVS compatible” (Casaba Watcher, CodeScan for Visual Studio) ale nie pokrywają pełnego zakresu ASVS

ASVS nie definiuje sposobu testowania

- Definiuje zakres sprawdzeń
- Do testującego należy sprawdzenie danego zabezpieczenia

Sprawdzenia nie są wyczerpujące

- Np. brak podatności w logice aplikacji



OWASP Testing Guide

://www.owasp.org

Zbiór wskazówek na temat testowania bezpieczeństwa aplikacji

Raczej encyklopedia / leksykon niż standard audytowy (349 stron)

Opis 66 „security controls”, dla każdej:

- Brief summary
- Description of the issue
- Black box testing and examples
- Gray box testing and examples
- References

To nie są dokładne instrukcje ale wskazówki, pomysły, przykłady !

Wskazówki do pisania raportu

- Propozycja metodyki oceniania wpływu podatności na ryzyko

Rozdziały Testing Guide nie mapują się bezpośrednio na sprawdzenia ASVS



Podsumowanie

Testy bezpieczeństwa są niezbędne dla większości aplikacji jako minimum weryfikacji bezpieczeństwa

Ważne jest dobre zdefiniowanie zakresu i sposobu wykonania testów

- Analiza zagrożeń
- Jakiego rodzaju badań?

Sprowadzenie ofert do jednego mianownika

- Można użyć OWASP ASVS i określić poziom weryfikacji
- ASVS nie definiuje sposobu przetestowania
- Ważne jest doświadczenie i rzetelność testera

Pamiętajmy, że produktem jest raport a nie sam test



Dziękuję za uwagę



wojciech.dworakowski@securing.pl

Tel.: 12 4252575

<http://www.securing.pl>