Advanced session stealing (část 2.)

Vložil/a c0stra [1], 6 Červen, 2006 - 16:46

- Hacking [2]
- Hacking method [3]
- Security [4]

Před nějkým časem jsem uveřejnil článek, který se zabýval kradením webových session. Tehdy mi unikl jeden dost podstatný typ útoku. Vzhledem k tomu, že jeho závažnost považuji bez nadsázky za alarmující, cítím povinnost na toto nebezpečí upozornit a doufám, že se administrátoři webových služeb zařídí minimálně podle rad, které uvedu na závěr.**Od: c0stra**

Nebudu zde znovu vysvětlovat celý princip získávání cizích session. Pro plné pochopení tohoto textu je však nutné přečíst i předchozí článek Advanced session stealing (část 1.) [5].

Obsah

1. Útok zevnitř
    1.1 Webová aplikace
    1.2 Buffer overflow
    1.3 Potřebné předpoklady
    1.4 Kontrola nad session
2. Spojení s jiným typem útoku
    2.1 Obcházení překážek
    2.2 Další fáze
3. Zabezpečení
    3.1 Striktní kontrola parametrů
    3.2 Zdravá paranoia
    3.3 Stavový automat
    3.4 D.M.Z.
4. Užitečné odkazy

1. Útok zevnitř

1.1 Webová aplikace
1.1 Buffer overflow
1.2 Potřebné předpoklady
1.3 Kontrola nad session

Typ útoku, který zde budu popisovat, je vlastně útok uvnitř původní napadené session. O co přesně jde vysvětlím v následujících odstavcích. Je ale potřeba zdůraznit, proč je tento typ útoku tak nebezbečný. Odpověď je jednoduchá. Zatímco jakémukoliv útoku z předchozího článku se dalo zabránit jednoduše kontrolou IP adresy, tady tomu tak **NENÍ**. Je to logické, protože útok probíhá přímo z IP adresy napadené session.

1.1 Webová aplikace

Nejdříve se musíme trochu podívat na to, jak vlastne taková webová aplikace

funguje. Záměrně zde poižívám pojem "aplikace", kterým mám na mysli službu, která vám umožní přes webové rozhraní provádět různé operace.

V minulém článku jsem rozebíral, jakým způsobem a jakými prostředky se provádí ověřování platnosti session. Tentokrát se stejným způsobem podíváme na to, jak funguje interakce mezi uživatelem a aplikací při provádění různých operací. Komukoliv trochu zběhlému v těchto věcech je jasné, kam mířím.

Jakákoliv akce ve webové aplikaci se provádí na základě zadaných parametrů zavoláním určitého výkonného skriptu. URL tohoto skriptu je obsaženo buď v referenci odkazu, nebo ve vlastnosti "action" u formuláře. Na předávání příslušných parametrů jsou k dispozici metody GET a POST protokolu HTTP. Metodu, která se používá poznáte snadno. Pokud se jedná o odkaz, jde vždy o metodu GET a pokud se jedná o formulář, je metoda určena vlastností "method". Když tato vlastnost u formuláře chybí, je použita metoda GET.

Pro nás bude především důležitý rozdíl mezi temito metodami. Ten spočívá v tom, že metodou GET se posílají parametry zakódované přímo v URL (viz. předchozí článek), zatímco u metody POST se parametry posílají až v contentu požadavku.

Příklad metody GET

```
GET /follow/rabbit.php?he=is&the=one HTTP/1.1
Host: <a href="http://www.matrix.org<br />
User-Agent:" title="">www.matrix.org<br />
User-Agent:</a> Mozilla/5.0 (compatible; Konqueror/2.2; FreeBSD)
Authorization: Basic:YWRtaW46dGFqbmVoZXNsbwo=
Accept: text/*, image/jpeg, image/png, image/*, */*
Accept-Encoding: x-gzip, gzip, identity
Accept-Charset: Any, utf-8, *
Accept-Language: en, cs_CZ, cs_CZ.ISO_8859-2
Referer: <a href="http://matrix/opennix/session.php?sid=sedtgIUrvhgjhtFhtf587gf<br />
Connection:" title="http://matrix/opennix/session.php?sid=sedtgIUrvhgjhtFhtf587gf<br
/>
Connection:">http://matrix/opennix/session.php?sid=sedtgIUrvhgjhtFhtf587gf<br />
Connection:</a> Keep-Alive
```

Příklad metody POST

```
POST /follow/rabbit.php HTTP/1.1
User-Agent: Opera/6.1 (FreeBSD 4.7-STABLE i386; U)  [en]
Host: <a href="http://www.matrix.org<br />
Accept:" title="">www.matrix.org<br />
Accept:</a> text/html, image/png, image/jpeg, image/gif, image/x-xbitmap, */*
Accept-Language: cs
Accept-Charset: windows-1252, utf-8;q=1.0, utf-16;q=1.0, iso-8859-1;q=0.6, *;q=0.1
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0
Referer: <a href="http://www.matrix.org/follow/rabbit.php<br />
Connection:" title="http://www.matrix.org/follow/rabbit.php<br />
Connection:">http://www.matrix.org/follow/rabbit.php<br />
Connection:</a> Keep-Alive, TE
TE: deflate, gzip, chunked, identity, trailers
Content-type: application/x-www-form-urlencoded
Content-length: 13

he=is&the=one
```

Důležitá věc však je, jak k parametrům zaslaným rozdílnými metodami přistupuje aplikace, tedy skript, který má data zpracovat a provést požadovanou akci. Ten buď rozlišuje metody a to co má dostat GETem prostě z POSTu nevezme a naopak (to především), nebo metody nerozlišuje a parametry přijímá z obou metod. Protože obě metody se nedají kombinovat, je druhá varianta velmi pohodlná v případě, že máte nějaké parametry, které musí být přítomny v každém požadavku, ať se jedná o jakoukoliv metodu (typicky pokud máte session postavenou na identifikačním parametru).

1.2 Buffer overflow

V minulém článku jsem popisovaný útok přirovnal k trojskému koni. Tento útok se nápadně podobá klasickému útoku "Buffer overflow". Útočník se snaží přinutit systém (v tomto případě webovou aplikaci) vykonat vlastní instrukce, což dosáhne tím, že vloží instrukce mezi vkládaná data (parametry) tak, aby se vykonaly. Problematikou přesování návratové adresy funkce na stacku se tu zabývat nebudu, případní zájemci o tuto problematiku můžou přeskočit na "užitečné" odkazy.

V našem případě se jedná o umístění vhodného HTML kódu na takové místo, aby ho browser napadeného uživatele interpretoval a vykonal v aplikaci naše "instrukce" zadané ve formě vhodných URL s potřebnými parametry.

Hlavní rozdíl oproti "kradení session" je v tom, že tentokrát se nejedná o URL na náš externí skript, ale jedná se o vnitřní URL aplikace s nastavenými parametry akce, ke které chceme aplikaci donutit.

1.3 Potřebné předpoklady

Samozřejmě zase nemůžeme čekat, že půjde napadnout jakákoliv služba. Dokonce by se dalo říct, že předpoklady na takový útok jsou o něco specifičtější, ale možná byste byli překvapeni, kolik služeb tyto předpoklady splňuje. Dokonce takto napadnutelných služeb je více než pouze na session stealing.

Nejlepším předpokladem je, pokud se dá do služby vložit přímo HTML kód, který potřebujeme. Typicky jsou to přílohy v e-mailech, ale je dobré zkoušet cokoliv, jakmile někde máte formulář, jehož obsah se po odeslání někomu zobrazí v jeho aplikaci, je tu šance, že bude ošetření chybět. V tomto typu útoku není zrovna optimální vkládat požadovanou adresu jako odkaz. Jednak je přímo vidět, co se děje (i když někteří uživatelé mohou klidně na takový odkaz kliknout s klidným svědomím, že se jedná o adresu toho "známého" serveru, kde právě jsou přihlášeni) a jednak zde není možnost maskování akce. Navíc takový způsob znamená pouze jednorázovou akci a okamžitou ztrátu kontroly.

Ideální případ je, když vám aplikace umožní vložit (jakýmkoliv způsobem) např. několik tagů <img scr="" heigth=0 widht=0> s vámi zadanými URL obrázků zasebou. Je samozřejmě možné použít i jiné tagy, které si dotahují obsah z externího URL, ale nepřesměrují se na něj a uživatel zůstane stále na aktuální stránce, a pokud možno si ničeho nevšimne (např. <link rel="stylesheet" type="text/css" href="">, nebo <script type='text/javascript' src=''></script>).

Někdy se může situace trochu zkomplikovat. Předchozí podmínky stačí v podstatě jen na dva specifické typy session. Cookie based a session, které mají identifikaci přímo v cestě. Pokud máte aplikaci, která je postavena na posílaném parametru, je zapotřebí, aby aplikace umožnila interpretaci JavaScriptu, protože je třeba z aktuálního URL vyseparovat identifikační řetězec, případně další parametry.

CC-BY-SA Security-Portal.cz | secured by paranoid sense | we hack to learn

Co je ale velmi důležité, je to, že není nutným předpokladem to, že aplikace nekontroluje IP adresu session. Vzhledem k tomu, že útok probíhá z IP adresy přímo napadeného uživatele, taková kontrola není překážkou.

1.4 Kontrola session

Tento typ útoku neumožňuje přímo převzít kontrolu nad session. Sám o sobě pouze umožňuje provést určitý počet akcí uvnitř session, případně po sobě zamést stopy, dát někam notifikaci o provedení a to je vše. Může však např. připravit kompletní převzetí a pak ho iniciovat. O tom budou další kapitoly.

Jěště jeden podstatný předpoklad je třeba vzít v úvahu, který jsem zatím úmyslně zamlčel, ale asi pozorným čtenářům už vrtá hlavou. Je totiž zřejmé, že akce, které potřebujeme vykonat musí být proveditelné pouze přes metodu GET. Pokud aplikace běží na PHP, je velká pravděpodobnost, že i data normálně posílaná POSTem můžete předat i GETem, a výsledek bude stejný. Nicméně je potřeba vždy ověřit to co chcete vykonat.

Jak jsem uvedl, tento útok neumožňuje převzetí kontroly nad session. Umožňuje však jiné věci. Jen pro představu je možné např. přeposlat důležité yprávy z emailového účtu, nebo jednodušeji nastavit přeposílání veškeré příchozí pošty, případně nastavit uživatelské údaje pro komunikaci s administrátorem služby, dokonce mě překvapilo, když jsem se setkal se službou, kde bylo uvnitř session možné změnit si heslo bez zadání původního! Asi nemusím zdůrazňovat, že to může vést ke kompletnímu převzetí účtu.

2. Spojení s jiným typem útoku

2.1 Obcházení překážek
2.2 Další fáze

Když jsem tento typ útoku odhalil, bylo to právě jako přípravná část útoku na ukradení session. Vzhledem k poměrně velké omezenosti a malé možnosti kontroly úspěšnosti, najde asi tento "Buffer overflow" větší uplatnění jako součást, či příprava nějakého komplexnějšího útoku. Může například změnit nastavení a tím umožnit (popř. i inicializovat) útok jiného typu.

2.1 Obcházení překážek

Typickým využitím může být "násilné" (případně i dočasné) splnění předpokladů pro jiný útok.

Dost často má uživatel (bohužel) možnost sám si určit úroveň zabezpečení dané aplikace. Například si sám může zvolit, jestli chce používat kontrolu IP adresy. Ta se tím pádem nastavuje někde v aplikaci formulářem, nebo pouze kliknutím na nějaký link. Není tedy nic jednoduššího, než ověřit, jestli je daná operace přístupná metodou GET a kontroly se zbavit.

Tímto způsobem se samozřejmě dají obejít i jiné překážky. Pokud například k něčemu potřebujete znát kontrolní otázku a odpověď, prostě jí nastavíte na nejakou, kterou znáte.

2.2 Další fáze

Důležitou možností je také schopnost inicializovat další fázi útoku. Pouze při tom nesmíme zapomenout na synchronizaci. To že v HTML kódu jsou dva

CC-BY-SA Security-Portal.cz | secured by paranoid sense | we hack to learn

obrázky za sebou rozhodně neznamená, že se budou načítat postupně. Browser se obvykle snaží dotáhnout všechny externí součásti (obrázky, styly, zdrojové kódy js), dotahovat parlelně. I kdyby to tak ale nedělal, nemáte nikdy zaručeno pořadí ani to, že zdroj, který se začal dotahovat dřív, bude komplet načten dřív než další začne.

Není tu bohužel efektivní způsob synchronizace, nějaká signalizace, která by odstartovala další krok. Je však pravděpodobné, že pokud se další krok spustil před dokončením předchovího, obdrží nějakou signalizaci chyby od aplikace (např. místo otevřené session dostane login). Toto už se dá nějakým způsobem testovat. Je tedy třeba útok v určitých časových intervalech útok opakovat a testovat tuto signalizaci, dokud nedostaneme co potřebujem, nebo nepřekročíme určitý čas (jinak by se mohlo stát při neúspěšné předchozí fázi, že nám druhá fáze poběží do nekonečna, což by mohlo být podezřelé administrátorům).

3. Zabezpečení

3.1 Striktní kontrola parametrů
3.2 Zdravá paranoia
3.3 Stavový automat
3.4 D.M.Z.

V předchozím článku jsem zdůrazňoval především ochranu pomocí kontroly IP adresy. Jak se ale ukázalo, je pouze tato ochrana nedostačující. To ale neznamená, že není potřeba. Právě naopak. Tuto kontrolu je stále potřeba provádět, a především chci apelovat na striktní nastavení této ochrany bez možnosti ji vyřadit. Nechat neznalého uživatele vybírat úroveň zabezpečení je jedno z největších rizik. Navíc je tu riziko přenastavení úrovně zabezpečení právě tímto útokem. Striktní kontrola IP adresy je pouze začátek a tentokrát už nedostačující.

3.1 Striktní kontrola parametrů

Každý skript, ať už se jedná o CGI, php, asp, jsp či servlet pracuje s určitými parametry. Problém skriptovacích jazyků je v tom, že tyto parametry není třeba nijak deklarovat, a že obvykle nemají určen datový typ. Při vývoji se pak snadno stane, že se nový parametr zavede kdekoliv ve skriptu, jeho použití se dokonale neošetří. Program prostě očekává určitou hodnotu, ale může se stát, že při vložení nějaké nepředvídané hodnoty se bude chovat úplně jinak.

Na této úrovni by mělo být zabezpečení, které před jakýmkoliv použitím nejprve otestuje všechny parametry, které do aplikace přišly. To znamená kontrolovat, jestli proměnná přišla správnou metodou, jestli má správný formát a případně vyfiltrovat všechny nebezpečné prvky (např. převést všechny znaky na HTML entity).

Rád bych zde zdůraznil že právě použitím metody POST a striktní kontrolou toho, že data byla skutečně postem poslána, je významným zvýšením úrovně zabezpečení, přesto že uživatele často straší při odesílání dat touto cestou varovné okno prohlížeče.

3.2 Zdravá paranoia

Samozřejmě často máte aplikaci, do které vstupují ještě externí data. V takovém případě je potřeba tato data testovat stejným způsobem, jako

parametry od uživatele, nebo spíš ještě víc. Taková data byse měla pořádně protestovat regulárními výrazy na očekávaný formát a opět odfiltrovat všechny nebezpečné prvky.

3.3 Stavový automat

Poněkud vyšší stupeň paranoii se dá dosáhnout implementací stavového automatu nad aplikaci. Ten může být buďto kompletní, postihující všechny stavy aplikace, nebo pouze částečný, který postihuje pouze stavy důležité pro některé operace. Nabízí se možnost implementovat tento automat na straně serveru, ale jednak by omezil session na jedno okno prohlížeče (pokud by uživatel otevřel druhé okno, pravděpodobně by se v tom starším později dozvěděl, že dělá něco nelegálního), a hlavně by výše popsanému útoku nezabránil, ten by v takovém případě musel mit pouze víc instrukcí.

Implementace takového automatu by mohla být taková, že každý stav by měl svůj hash, který bude pro každou session jiný, pokud možno neodvoditelný (snadno) z jakýchkoliv informací, které má útočník k dispozici. Takový hash by se tedy poslal vždy (nebo pouze při důležité události) ke klientovi a ten by ho s dalším požadavkem poslal zpět. Na servru by se pak vyhodnotil zaslaný stav na základě požadované akce. Pokud by tato kombinace stav - požadavek neexistovala, byl by požadavek vyhodnocen jako útok. Je však samozřejmě potřeba, aby se hash stavu potřebný pro klíčové akce nedal zjistit z místa, které je potencionálně napadnutelné.

3.4 D.M.Z.

Někdy je potřeba zobrazit v aplikaci externí data beze změn. Typicky přílohy e-mailů na webmailech. Kvůli možnosti stažení nebo kdovíčemu jinému je potřeba dát taková data ven bez jakýchkoliv změn. Taková data samozřejmě automaticky znamenají veliké nebezpečí. Jak jsme se již přesvědčili, kontrola IP adresy nepomůže. Řešení takovéto situace je trochu podobné předchozímu automatu, ale je trochu jednodušší.

Pojem demilitarizované zóny asi zná každý (i když možná né každý má představu o tom co to ve skutečnosti je). Zjednodušeně řečeno je to zóna, která má z různých důvodů zmírněná bezpečností pravidla "z venku" a zvýšená pravidla "dovnitř". Jinými slovy pokud se podaří např. kvůli nejakým povoleným službám nabourat počítač v DMZ, nesmí být možné se z něj dostat do vnitřní sítě. Na podobném principu funguje následující opatření.

Pokud se v aplikaci vyskytují uvedená nebezpečná neošetřená data, je potřeba pro ně vytvořit takovou demilitarizovanou zónu. To znamená vlastně je vystrčit z aktuální session a odkazovat se na ně pomocí unikátního identifikátoru (nějakého hashe), ze kterého nelze (efektivně) odvodit session ID, ale pro který je na straně serveru jasně dané, ke které session patří.

Například pro přílohy e-mailů se může při každém přihlášení uživatele vygenerovat tabulka hashů pro jednotlivé přílohy. Uživatel je pak získá pouze přes tento hash (pokud možno ještě po přesměrování, aby se původní session nedala zjistit ani z hlavičky Referer. Uživatelův browser pak přílohu otevře a interpretuje. V tom momentě proběhne útok, který má ale k dispozici pouze autorizační parametry naší DMZ, se kterými může napadnout tak akorát sám sebe :o), protože do session se nedostane.

4. Užitečné odkazy

CC-BY-SA Security-Portal.cz | secured by paranoid sense | we hack to learn

Tentokrát zde neuvádím žádné ukázky skriptů. Není potřeba nic nového, protože techniky jsou stejné, takže stačí si projít ukázky v předchozím článku.

Nebudu zde ani uvádět služby, které jsou na tento útok vulnerabilní, protože u některých by to mohlo bez nadsázky znamenat přímé ohrožení všech uživatelských účtů! Rád bych tedy doporučil především **administrátorům** a **vývojářům** takových služed, aby své produkty důkladně prověřili.

Advanced session stealing (část 1.) [6] - předchozí článek o kradení webových session

http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci549024,00.html [7]

http://packetstorm.decepticons.org/ [8] - výborný server co se týká jakýchkoliv bezpečnostních záležitostí

http://www.insecure.org/ [9] - pokud hledáte nástroje na bezpečnostní audit serveru či sítě

Web autora článku: http://sg1.homeunix.org/ [10]

URL článku: https://security-portal.cz/clanky/advanced-session-stealing-%C4%8D%C3%A1st-2

Odkazy:
[1] https://security-portal.cz/users/c0stra
[2] https://security-portal.cz/category/tagy/hacking
[3] https://security-portal.cz/category/tagy/hacking-method
[4] https://security-portal.cz/category/tagy/security
[5] http://www.security-portal.cz/clanky/advanced-session-stealing-cast-1.html
[6] http://www.security-portal.cz/clanky/advanced-session-stealing-cast-1
[7] http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci549024,00.html
[8] http://packetstorm.decepticons.org/
[9] http://www.insecure.org/
[10] http://sg1.homeunix.org/

CC-BY-SA Security-Portal.cz | secured by paranoid sense | we hack to learn