PV177 - Útoky na vlastní stroj

Lukáš Ručka (učo: 359687), Jan Doleček (učo: 256331)

5. května 2010

1 Zadání

Vyzkoušet si různé metody útoku proti distribuci DVL (Linux). Sepsat zprávu.

2 Lokální útoky

Pro provedení tohoto typu útoku je zapotřebí mít základní práva k systému (tzn. např. uživatelský shell). K tomu se může útočník dostat pomocí útoků na síťové služby. Obvyklým cílem útoku je zvýšení oprávnění účtu, který hacker ke svým operacím používá, avšak může se jednat i o pokus napadnout nějakou službu pomocí její známé nebo čerstvě objevené slabiny. V další části je zmíněno několik vyzkoušených a již nějakou dobu známých útoků vůči slabinám programů umístěných v systému.

2.1 Útok využívající volání jiného programu

Mějme hypotetickou situaci, kdy máme uživatele A a program P (s právy L, patřící uživateli B s nastaveným SUID bitem) a útočníkův kód Q.

Uživatel A spustí program B, který během svého provádění, například na základě proměnné prostředí, volá program R. Jelikož má tento program P nastavený SUID bit, je spuštěn jako proces s právy vlastníka a proto i jeho dceřiné procesy, pokud se této výhody dobrovolně nevzdají, pokračují s těmito oprávněními. Jako příklad takového postupu může být program, který si obstarává stránkování pomocí programu, uvedeném v proměnné prostředí $ { PAGER} . Toho lze docílit například tak, že si založíme pojmenovanou rouru, do které z programu přesměrujeme výstup, a ze které bude číst právě stránkovací program. Takový stránkovací program je pak při nesprávné implementaci tohoto mechanismu spuštěn s oprávněními L. Vzhledem k faktu, že se SUID bit obvykle používá pro přístup na vyšší úroveň oprávnění, může takto útočník eskalovat svá oprávnění (byť třeba v omezené míře).

Známým případem tohoto druhu útoku je exploit proměnné $ { PAGER} u programu cdrecord ze září roku 2004[1]. Tato konkrétní bezpečnostní díra se již v DVL nevyskytuje.

2.2 Útok využívající přetečení bufferu

Tento druh útoku se opírá o to, že někdy není kontrolován vstup programu, zda nepřesahuje velikost oblasti paměti, kam je vstup nahráván. Toho může využít útočník tak, že svým vstupem přepíše návratovou adresu uloženou na zásobníku tak, že nová návratová adresa bude ukazovat do jeho vstupu. Ten pak může obsahovat předchystaný kód, který je spuštěn pod oprávněním dané aplikace. Takový kód má obvykle podobu tzv. nopsledu + útočných instrukcí. Nopsled je slangový výraz pro posloupnost instrukce nop [2] (No OPeration).. Za touto posloupností již následují vlastní instrukce útoku. Procesor tedy při návratu z rutiny najde adresu ukazující do nopsledu, ten přeskočí a spustí útočníkův kód.

Toto je popis konkrétní verze techniky, využívající přetečení dat na zásobníku. Cílem útočníka nemusí být jen spuštění kódu, nýbrž i manipulace s daty, nad nimiž program operuje. To je výhodou oproti předchozímu uvedenému útoku.

Možné obrany proti tomuto útoku jsou:

- Ošetřovat délku jakéhokoli vstupu

[1] http://www.milw0rm.com/exploits/438

[2] Tato instrukce je jednobajtová a je procesorem ignorována. V běžné praxi se používá na zarovnání spustitelného kódu na adresy dělitelné počtem bitů architektury daného stroje (na i?86 platformách se obvykle jedná o 4B nebo 8B pro 64bit architektury)

- Metoda "kanárků"

  Tato metoda je pojmenována podle praxe horníků, kteří když potřebovali vědět, zda je v šachtě dýchatelný vzduch, spustili do ní klícku s kanárkem. Pokud kanárek přežil, bylo vše v pořádku. Jedna z možných implementací spočívá v alokaci místa na vrcholu zásobníku a uložení nějakého magického čísla do tohoto prostoru. Pokud při opouštění volané rutiny najdeme neplatnou hodnotu, máme jistý pokus o útok a skončíme. Obecně se tedy jedná o zajištění integrity dat na zásobníku během průběhu zranitelné funkce. (kompilátor gcc umí tuto kontrolu vkládat automaticky)

- Randomizace prostoru virtuálních adres

  Aby útočník mohl úspěšně přepsat návratovou adresu, potřebuje k tomu předem znát adresu paměti, kde je uložen jeho kód. Stačí mu program spustit s náhodnou návratovou adresou, což však způsobí pád programu. V záznamu o pádu programu si pak utočník může najít adresu, kde byla provedena neplatná operace, a tím adresu svého kódu zjistit. Proto vznikla technika náhodného mapování kódu a knihoven do prostoru adres virtuální paměti. Při každém spuštění programu operační systém napamuje bloky kódu a dat na jiné virtuální adresy a tudíž útočník nemůže předpovědět správnou návratovou adresu, která by odkazovala na jeho kód. Tento typ útoku se tak stává nepoužitelným.

Po shlednutí demonstrace samotného útoku[3]

Tip: Na konec vlastních útočných instrukcí je lepší přidat instrukce pro zápis původní návratové hodnoty do napadené oblasti paměti a skočit na tuto adresu. Vyhneme se tak například zaznamenání pádu hostitelského programu.

3 Vzdálené útoky

3.1 Remote Code Injection

PhpMyAdmin je program napsaný v jazyce PHP, který umožňuje jednoduché spravování databází mySQL v prostředí webového rozhraní. Pro jeho snadnou instalaci je k archivu programu přibalen i konfigurační průvodce, který tento program nastaví, a umožní tak snadnou instalaci komukoliv. Průvodce funguje tak, že se zeptá na potřebné údaje k databázi (případně se některé pokusí dohledat sám), a při dokončování vytvoří konfigurační soubor nesoucí zadané nastavení a uloží jej na disk. Program phpMyAdmin z tohoto souboru načítá konfiguraci. Jak je u PHP aplikací zvykem, konfigurační soubor je klasický spustitelný .php soubor, ve kterém jsou inicializované konfigurační proměnné. Protože verze phpMyAdminu instalovaná na strojích DVL nekontroluje důsledně vstup zadaný uživatelem, je možné uměle vytvořit http požadavek se specifickými daty, které znetvoří konfigurační soubor a přinutí jej tak provádět akce, které sami chceme. Konkrétně se jedná o řádek s kódem:

$ cfg['Servers'][$ i]['host'] = 'hodnota zadana uzivatelem' ;

Pokud uživatel zadá jako vstup např.:
'; skodlivy-kod; //
vznikne v konfiguračním souboru řádek:

$ cfg['Servers'][$ i]['host'] = ' ' ; skodlivy-kod; //' ;

Při spuštění tohoto souboru se nyní nejen provede inicializace proměnných, ale také se spustí libovolný kód zadaný útočníkem se stejným oprávněními, jako má spuštěná PHP aplikace. Je tak možné získat shell na stroji, na který nemáme vůbec přistup. Případně útočník může zkusit použít zranitelnosti popsané ve 2. kapitole pro eskalaci práv a pokusit se tak dostat na shell administrátora. (pomocí příkazů v PHP stáhnout a spustit externí program, který umožní úročníkovi připojit se na shell administrátora) Při spuštění tohoto souboru se nyní nejen provede inicializace proměnných, ale také se spustí libovolný kód zadaný útočníkem se stejným oprávněními, jako má spuštěná PHP aplikace. Je tak možné získat shell na stroji, na který nemáme vůbec přistup. Případně útočník může zkusit použít zranitelnosti popsané ve 2. kapitole pro eskalaci práv a pokusit se tak dostat na shell administrátora. (pomocí příkazů v PHP stáhnout a spustit externí program, který umožní úročníkovi připojit se na shell administrátora)

(Pozn.: na strojích DVL nebyl konfigurační průvodce použit, nýbrž byla konfigurace vytvořena jinde a pouze překopírována. Konfigurátor tak neměl nastaveno oprávnění zapisovat a nemohl zranitelný soubor vytvořit. Zde se tedy útok nepovedl.)

4 Závěr

Pomocí několika postupných útoků může útočník relativně snadno získat administrátorská práva k systému, ke kterému by neměl mít vůbec žádný přístup. Velmi často za to může trestuhodná nedbalost či lenost programátorů (označené místo okomentují, že je třeba později toto ošetřit, a samozřejmě to nikdy neudělají) při ošetřování vstupu zadaného uživatelem.

[3] http://www.youtube.com/watch?v=ZZ0LVAFIDrA