AN ATTACK ON THE LINUX SYSTEM CALL

Boris Procházka
Master Degree Programme (2), FIT BUT
E-mail: xproch63@stud.fit.vutbr.cz

Supervised by: Tomáš Vojnar
E-mail: vojnar@fit.vutbr.cz

ABSTRACT

This paper deals with methods of Linux kernel hacking on the Intel Architecture. It presents a classification of attacks and their impacts. Further, the paper discusses the Linux system call mechanism and proposes an original attack on this mechanism based on execution flow hijacking.

1 ÚVOD

Útoky na výpočetní prostředky vznikaly, existují a vyvíjí se stejně jako výpočetní prostředky samotné. Moderní počítačové systémy řídí komplexní operační systémy, mezi které můžeme počítat i Linux. Linux je znám svojí politikou otevřeného zdrojového kódu, která umožňuje jádro studovat [1, 2], měnit a zvyšovat jeho bezpečnost. Tento článek se zaměřuje na zranitelnosti rozhraní systémových volání v jehož rámci popisuje nový způsob bezpečnostní hrozby.

2 KLASIFIKACE ÚTOKŮ

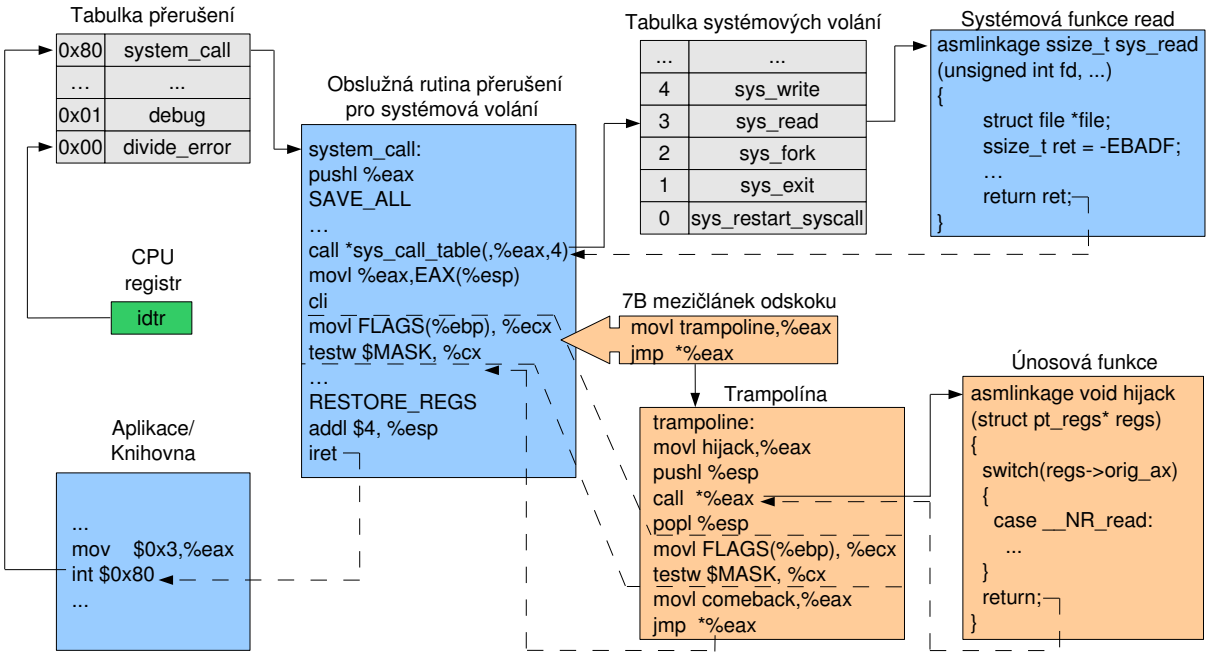
Útoky na operační systém Linux lze klasifikovat do dvou základních tříd [3]. Jedná se o útoky na:

1. **uživatelský režim.** Do této skupiny patří nahrazení administrátorských utilit (`ls`, `ps`, `...`) a systémových knihoven vlastními verzemi, které skrývají útočníkovy aktivity. Tento způsob útoku s sebou přináší nevýhodu modifikace binárních souborů, jejichž integrita je snadno ověřitelná přes kontrolní součty. Administrátor systému má dále možnost používat privátní sadu nástrojů z ověřeného zdroje (např. CD), která může být zkompilována staticky (nehrozí tak jejich infekce sdílenou knihovnou).

2. **jaderný režim.** Jde o útoky na jaderný paměťový prostor, především na rozhraní systémových volání a virtuální souborový systém. Hlavní výhoda útoku spočívá v jeho dopadu na celý systém, neboť jaderné služby jsou využívány všemi uživatelskými programy (`open`, `read`, `write`, `...`). Modifikací služeb můžeme výsledky těchto volání upravovat a uživatelským programům zpřístupnit pouze podmnožinu všech informací. Útoky na jaderný paměťový prostor jsou vysoce nebezpečné, neboť v případě úspěšné kompromitace stroje neexistuje spolehlivý způsob, jak útočníka odhalit (lze pouze předpokládat, že útočník nezamaskoval veškeré vedlejší efekty skrývaných prostředků – např. reakci skrytého procesu na signál.).

3 NAPADENÍ ROZHRANÍ SYSTÉMOVÉHO VOLÁNÍ

V dalším textu se zaměříme pouze na útok na rozhraní systémového volání. Schéma průběhu systémového volání zachycuje (bez oranžové části) obrázek 1. Aplikace či knihovny používají pro vyvolání systémových služeb softwarové přerušení `int $0x80`. K identifikaci systémové služby slouží registr `eax`, ostatní registry jsou využívány jako parametry. Vyvolání přerušení způsobí výjimku a následné přepnutí procesoru do jaderného režimu. Z tabulky přerušení, jejíž začátek určuje procesorový registr `idtr`, je zavolána obslužná rutina přerušení pro systémová volání. Ta uloží všechny registry, provede kontrolní operace a registr `eax` použije jako index do tabulky systémových volání k zavolání systémové funkce. Po jejím dokončení následují další kontrolní operace (tyto již významné z pohledu kapitoly 4) a návrat do uživatelského režimu instrukcí `iret`. Návratová hodnota je vrácena v registru `eax`.



Obrázek 1: Schéma únosu systémového volání

Práce [3] popisuje nejznámější metody napadení tohoto rozhraní, jež jsou cíleny především na tabulkové ukazatele nebo tabulkové záznamy. Jedná se o útoky na: registr `idtr`, záznam v tabulce přerušení, tabulkový ukazatel v obslužné rutině přerušení, záznam v tabulce systémových volání a útok odskočením z prologu systémové funkce.

4 METODA MODIFIKACE KÓDU OBSLUŽNÉ RUTINY PŘERUŠENÍ

Ve zbývající části textu se budeme zabývat myšlenkou modifikace části kódu obslužné rutiny přerušení. Naším cílem bude pozměnit průběh výpočtu tak, abychom mohli ovlivnit hodnoty vracené zpět do uživatelského režimu. Tomuto účelu nejlépe vyhovuje pokus o odskočení z funkce pomocí instrukce `jmp`. Při hledání vhodné části kódu pro modifikaci musíme zohlednit následující skutečnosti:

- **kód musí zůstat platný** – při přepisování kódu musíme zachovat platnost strojového kódu. V případě, že modifikací vznikne neplatná instrukce, vyvolá procesor výjimku.

Při přepisování kódu tedy musíme respektovat začátky a konce dalších instrukcí a nepřepisovat kód, který obsahuje návěští.

- **alespoň 7B paměti** – právě tolik zabírají strojové instrukce pro naplnění registru adresou a jeho použití pro odskok. Pro nás tato podmínka implikuje nutnost přepisu dvou a více instrukcí.
- **modifikovaný kód musíme nahradit** – kód, jehož přepis použijme pro vlastní účely, nemůžeme ze systému beztrestně odstranit. Před návratem řízení do místa únosu musíme zajistit jeho nahrazení.

Zajištění výše stanovených podmínek není snadnou záležitostí. V případě únosu totiž musíme být schopni obnovit výpočetní stav procesoru do stejného stavu jako před únosem.

První myšlenkou by mohl být pokus o modifikaci úvodní části obslužné rutiny. Tato idea však naráží na problém s obnovením výpočetního stavu, neboť musíme nahradit veškerou část kódu, až po část vyvolávající obslužnou rutinu operace (abychom mohli její výsledky ovlivnit). Úvodní část funkce není navíc úplně konstatní v rámci celé řady `2.6.x` a obsahuje problematická nepřímá volání.

Můžeme se však pokusit odklonit výpočetní tok až po provedení obslužné rutiny operace a pokusit se zmodifikovat její výstupní hodnoty. Analýzou kódu bezprostředně po systémové operaci můžeme nalézt jeho *konstatní* část, která je pro naše účely vhodná. Jedná se o test příznaků aktuálně běžícího procesu (obr. 1, část pro mezičlánek), který může být proveden i později. Překladem si můžeme ověřit, že tato dvojice instrukcí zabírá právě 8B, což je dostačující pro provedení odskoku. Skok provedeme do předem připravené funkce `trampoline`, která zajišťuje zavolání únosové funkce `hijack` a obnovení kontextu výpočtu po jejím návratu. Únosová funkce má přes zásobník přístup ke všem návratovým parametrům, může z nich snadno odfiltrovat nežádoucí informace a ovlivnit výsledky systémových volání. Detail útoku je znázorněn na obrázku 1 oranžovou částí.

5 ZÁVĚR

Článek čtenáři představuje unikátní způsob napadení rozhraní systémového volání v operačním systému Linux. Funkčnost útoku byla experimentálně ověřena na jádrech řady `2.6.x`, čímž by tento typ útoku mohl být využit útočníky jako základ pro implementaci rootkitu (programu pro tvorbu kompromitovaného prostředí) s vysokou mírou utajení.

Poděkování: Tato práce vznikla částečně za podpory grantu VUT FIT, FIT-S-10-1 a specifického výzkumu MSM0021630528.

REFERENCE

[1] R. Love. Linux Kernel Development, Novell Press, Indiana 46240 USA, 2 edition, 2005. ISBN 0-672-32720-1.

[2] B. P. Daniel, Cesati Marco. Understanding the Linux Kernel, O'Reilly, USA, 3 edition, 2005. ISBN 0-596-00565-2.

[3] B. Procházka. Metody útoků na operační systém Linux, bakalářská práce, FIT VUT, Brno, 2008.