Zvýšenie bezpečnosti aplikácií s využitím automatickej transformácie

Tobiáš Smolka

xsmolka@fi.muni.cz

Fakulta informatiky
Masarykova univerzita
Brno, Česká republika

Abstrakt

Tento príspevok skúma použitie bezpečnostne orientovaných transformácii na zvýšenie bezpečnosti aplikácií na kryptografických čipových kartách. Špeciálne sa zameriava na automatické odstraňovanie existujúcich zraniteľností, pridávanie preventívnych ochranných mechanizmov a detekciu potenciálnych zraniteľností. Jeho hlavným prínosom je návrh a implementácia transformácií, ktoré umožňujú do existujúcich aplikácií automaticky pridať konkrétne ochranné mechanizmy (ochrana proti analýze postrannými kanálmi, ochrana proti indukcii chyby, kontrola behu aplikácie podľa modelu a detekcia použitia potenciálne zraniteľných programových konštrukcií).

Kľúčové slová: Čipové karty, Java Card, syntaktická analýza, programové transformácie.

1 Úvod

Kryptografické čipové karty sa dnes využívajú v množstve oblastí, kde sa kladie vysoký dôraz na bezpečnosť. Žiadna aplikácia na čipovej karte však nie je absolútne bezpečná. Je známe množstvo konkrétnych útokov a neustále sa objavujú nové. A aj keď výrobcom karty implementované hardvérové ochranné opatrenia môžu byť pomerne efektívne, pri tvorbe bezpečnej aplikácie môže byť vhodné nasadiť aj **softvérové ochranné opatrenia**. Oproti hardvérovým opatreniam nezvyšujú cenu karty, môžu byť aplikovateľné priamo výrobcom aplikácie a umožňujú pridávať nové opatrenia i po vydaní karty.

Ručná aplikácia softvérových opatrení však nemusí byť vždy jednoduchá. Často sú ochranné opatrenia pomerne zložité (napr. nahradenie všetkých výskytov zraniteľných konštrukcií), spôsobia zneprehľadnenie kódu alebo zanesenie nových implementačných chýb. Niektoré opatrenia sú mierené na konkrétne typy čipových kariet a pri prechode na iný hardvér už nemusia byť potrebné ani vhodné.

Bezpečnostne orientované transformácie poskytujú pomerne elegantný spôsob ako pridať do aplikácie ochranné opatrenia a pritom netrpia zmienenými neduhmi. Z pôvodného kódu aplikácie vygenerujú nový kód obohatený o bezpečnostné mechanizmy. Dokážu preto zvýšiť bezpečnosť aplikácie automaticky a konzistentným spôsobom. Ich ďalšou výhodou je, že znižujú nároky na odbornosť programátora, ktorý vďaka automatizácii už nemusí mať tak detailné znalosti z oblasti bezpečnosti čipových kariet.

Podobne ako refaktoring sú bezpečnostne orientované transformácie považované za štrukturálne programové transformácie, ktoré môžu byť automatizované bez nutnosti hĺbkového pochopenia funkcionality. Oproti refaktoringu však zachovávajú iba očakávané správanie a menia správanie aplikácie v prípade útoku. Ďalším rozdielom je aj to, že sa aplikujú vždy až pred uvoľnením aplikácie. Vývoj tak môže pokračovať na pôvodnom prehľadnom kóde, kým bezpečnostné mechanizmy sú pridané neskôr.

Transformácie môžu existovať ako na úrovni zdrojového kódu aplikácie, tak i na úrovni preložených inštrukcií. Výhodou transformácie na úrovni zdrojového kódu je možnosť auditu výsledného kódu a priame porovnanie pôvodnej a upravenej verzie. Transformácie na úrovni inštrukcií však môžu byť efektívnejšie a za istých okolností aj spoľahlivejšie[1].

Okrem výhod môže použitie transformácií priniesť aj niekoľko nevýhod. Môžu zvyšovať pamäťovú, či výpočtovú zložitosť aplikácie a nekvalitné transformácie môžu do kódu zaniesť nové chyby. Preto by mali byť transformácie parametrizovateľné (možnosť selektívne aplikovať transformáciu iba na kritické časti aplikácie), dostatočne testované a auditované. Niektoré opatrenia však môžu byť natoľko zložité, že pre ne nemusí byť možné transformáciu vôbec vytvoriť.

[1] U transformácií na úrovni zdrojového kódu je napr. nutné zabezpečiť, aby kompilátor svojimi optimalizáciami neodstránil pridané bezpečnostné mechanizmy.

2 Vytvorená sada transformácií

Na demonštráciu možností bezpečnostne orientovaných transformácií bola vytvorená sada transformácií, ktoré sa zameriavajú na niektoré vybrané zraniteľnosti dnešných čipových kariet s platformou *Java Card*. Sú voľne dostupné ako súčasť projektu *CesTa* [2]. Popis jednotlivých transformácií je možné nájsť v [2] a [3]. V ďalšom texte budú jednotlivé transformácie v krátkosti predstavené.

Prvá z transformácií je určená na ochranu proti **odberovej analýze**. Špecializuje sa na konkrétnu zraniteľnosť, ktorá umožňuje pomocou odberovej analýzy získať hodnotu rozhodovacej podmienky *if* na niektorých typoch čipových kariet [1]. Transformácia automaticky nahradzuje zraniteľné konštrukcie sémanticky ekvivalentnými, no bezpečnejšími konštrukciami, ktoré využívajú náhodnosť.

Úlohou druhej transformácie je zvýšiť odolnosť aplikácie voči **indukcii chyby** tým, že pridá do kódu dátovú redundanciu. Je to obecná preventívna transformácia a je určená pre všetky typy kariet. Ku každej premennej primitívneho typu automaticky vytvorí jej tieňovú kópiu a do aplikácie pridá kód, ktorý počas behu kontroluje, či nedošlo k útoku a či sa útočníkovi nepodarilo zmeniť hodnotu niektorej z premenných. Táto tieňová kópia obsahuje inverznú hodnotú premennej, aby aplikácia bola schopná detekovať aj situáciu, kedy sa útočníkovi podarí naraz nastaviť hodnotu celej skupine pamäťových buniek.

Tretia transformácia je určená na robustnú **kontrolu stavov** aplikácie podľa definovaného modelu (grafu prechodov medzi stavmi). Tento model je definovaný pomocou jazyka *DOT* a na jeho vizualizáciu je možné využiť nástroj *GraphViz*. Transformácia automaticky pridáva do aplikácie kontrolu aktuálneho stavu a kontrolu prechodu.

Kontrola aktuálneho stavu sa vykonáva pri vstupe do každej funkcie. Kontroluje, či v grafe prechodov existuje akákoľvek hrana z aktuálneho stavu, ktorej popis je zhodný s názvom danej funkcie. Kontrola prechodu sa vykonáva pri každej zmene stavu. Berie do úvahy starý a nový stav a kontroluje, či v grafe prechodov existuje hrana zo starého stavu do stavu nového.

Posledná z transformácií je určená na detekciu možných problémov s **atomicitou transakcií** platformy *Java Card*. Ak je v jednej transakcii premenná modifikovaná ako atomickými, tak i neatomickými operáciami, môže to viesť k nepredpokladaným výsledkom a ohroziť bezpečnosť aplikácie. Takéto transakcie však obecne nie je možné opraviť automaticky, pretože nie je jasné, ktorá z operácií by mala byť odstránená. Transformácia sa to preto snaží detekovať a upozorniť programátora.

3 Použitie

S využitím nástroja *CesTa* je možné vytvorené transformácie parametrizovať a pomerne pohodlne aplikovať na už existujúce aplikácie. Transformácie boli testované jednotkovými testami a aplikované na niekoľko reálnych aplikácií (softvérová implementácia *SHA 512*, aplet *JOpenPGPCard*, ukážkové aplikácie z *Java Card Development Kit*, bezpečné úložisko kľúčov pre *TrueCrypt*).

Transformované aplikácie fungovali korektne. Zvýšenie pamäťovej a výpočtovej náročnosti bolo priamo závislé na štruktúre aplikácie a parametroch transformácie. Najnáročnejšie sú z tohto pohľadu ochrany proti odberovej analýze (duplikácia každej rozhodovacej podmienky) a indukcii chyby (duplikácia premenných). Ich spoločná aplikácia spôsobila približne dvojnásobné zvýšenie pamäťovej i časovej náročnosti. Pri reálnom nasadení transformácií môže byť preto výhodné aplikovať transformácie len na tie miesta, ktoré sú z hľadiska bezpečnosti kritické a dodatočnú ochranu vyžadujú. Z testov však vyplýva, že dnešné čipové karty majú dostatok výkonu a pamäte, aby zvládli podobnú dodatočnú réžiu.

Referencie

[1] Kůr, J. a Smolka, T. a Švenda, P.: *Improving Resiliency of JavaCard Code Against Power Analysis*, 29-39, Santa's Crypto Get-Together 2009, Trusted Network Solutions, 2009.

[2] Lorenc, V. a Smolka, T. a Švenda, P.: *Automatic source code transformations for strengthening practical security of smart card applications*, 93-115, Sborník příspěvků z 36. konference EurOpen.CZ, 2010, `http://www.europen.cz/Anot/36/eo-1-10.pdf` (november 2010)

[3] Smolka, T.: *Zvýšení bezpečnosti aplikací na čipových kartách s využitím automatické transformace*, Diplomová práca, 2010, `http://is.muni.cz/th/172671/fi_m/` (november 2010)

[2] Viac informácií na `http://cesta.sourceforge.net/` (november 2010).