Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Diplomová práce

Bezpečnostní studie aplikace

Bc. Karel Rozhoň

Vedoucí práce: Ing. Tomáš Zahradnický, Ph.D.

18. května 2015

Poděkování

Tímto bych rád poděkoval svému vedoucímu diplomové práce, Ing. Tomáši Zahradnickému, Ph.D., za jeho trpělivost, cenné rady a pomoc při tvorbě této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 18. května 2015 .....

Odkaz na tuto práci

Rozhoň, Karel. *Bezpečnostní studie aplikace.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Práce se zabývá bezpečnostní analýzou masově využívané aplikace. Představuje svému čtenáři metody, které jsou během analýzy použity a ty pak aplikuje na danou aplikaci. Součástí této práce jsou modelové útoky na nalezené zranitelnosti a návrhy možných protiopatření.

Klíčová slova Počítačová bezpečnost, zranitelnost, bezpečnostní analýza

Abstract

This thesis provides a security analysis of an chosen application.It indroduces few basic methods of this security analysis and uses them on the chosen application. Another part of htis thesis are also examples of attacks to discovered vulnerabilities and list of possible recommendations, how to mitigate them.

Keywords Computer security, vulnerability, security analysis

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V poslední době si mnoho IT odborníků začíná uvědomovat důležitost zajištění počítačové bezpečnosti. Počítače totiž ovlivňují každodenní život každého z nás a nezáleží na naší profesi. Čím dál více z nás počítače využívá k provádění bezhotovostních plateb, které nám usnadňují život. Jedná se jen o jeden konkrétní příklad, ale zajisté si všichni uvědomujeme, že se jedná o operaci, u které je důležité zajistit naši bezpečnost. Tato bezpečnost je ovlivňována každým nástrojem, který do počítače instalujeme, a každou hardwarovou součástí kterou využíváme my i veškeré body na trase ke zmiňovanému serveru s internetovým bankovnictvím. Jak je naznačeno již v názvu této diplomové práce, budeme se v ní zabývat bezpečnostní analýzou volně dostupné aplikace. Pokusíme se tedy odhalit případná zranitelná místa v daném programu a prozkoumat tak bezpečnost jedné z komponent ovlivňujících počítačovou bezpečnost uživatele. Pokud by program obsahoval závažné nedostatky, které potenciální útočník snadno zneužije, pak by jeho používání mohlo znamenat vážné bezpečnostní riziko. V případě, že v rámci této práce objevíme nějakou zranitelnost, pokusíme se odhadnout dopad jejího eventuálního zneužití na uživatele. V takovém případě se pokusíme navrhnout možné opravy v návrhu či realizaci programu. Ty by vedly k minimalizování rizika zneužití této zranitelnosti a zároveň by i mohly posloužit jako základ pro případnou oficiální opravu vydanou vydavatelem softwaru. Protože však bude tato práce veřejně dostupná a my nechceme vystavit uživatele bezpečnostnímu riziku, které by mohlo jejím zneužitím nastat, anonymizujeme veškerá zkoumání, která v ní provedeme. Zkoumanou aplikaci budeme nazývat pseudonymem Charlie a v přiložených snímcích a datech z běhu programu odstraníme či zakryjeme informace, které by umožnily program identifikovat.

Program k prozkoumání nebyl dopředu určen, a proto jsme se rozhodli zvolit ho zejména na základě jeho rozšířenosti mezi uživateli. Naším cílem tedy bude prozkoumat bezpečnostní vlastnosti u takového programu, který je uživateli

poměrně hojně využíván a zároveň je zdarma dostupný již po mnoho let. Tím je téměř zaručeno, že se dostal do podvědomí široké masy uživatelů, což s sebou často přináší i falešnou důvěru v daný produkt. Uživatelé mohou nabýt dojmu, že takto zavedený a tedy tradiční program je bezpečnější než konkurenční produkt, který je na trhu relativně krátký čas. Zároveň pokud některému programu více věříme, pak se zpravidla tolik nezabýváme možnostmi jeho nastavení během instalace. I proto je pro nás zajímavé se na takový program zaměřit a detailněji prozkoumat, jak s důvěrou svých uživatelů v praxi zachází. Pokud bychom odhalili některé významné bezpečnostní nedostatky, pak by jejich potenciální zneužití znamenalo ohrožení poměrně velkého množství uživatelů.

V rámci této diplomové práce se nejprve seznámíme s jednotlivými kroky analýzy, které budeme provádět, a s nástroji, které nám tuto analýzu usnadní. Obecně se dá konstatovat, že nejvyšší riziko v používání libovolného programu znamená každý vstup do tohoto programu. Vstupem do programu zde myslíme například data, která programu poskytne uživatel v době jeho běhu. Nesmíme však opomenout ani další data, která se například s časem mění či závisejí na konkrétní konfiguraci operačního systému. Ty totiž vývojář softwaru během jeho vývoje nemůže znát, a proto je nutné je programu poskytnout v době jeho běhu. Zaměříme se tedy zejména na tyto možné vstupy do programu a na jejich zabezpečení. Například nás zajímá, kdo a za jakých podmínek může tyto data upravovat. Pokud se nám podaří naleznout takový vstup do programu, který lze relativně snadno podvrhnout, pak se zaměříme na část programu, která ho využívá, a na ověřování platnosti těchto dat. V této fázi se pokusíme odhalit nedostatečné mechanismy těchto kontrol. Cílem bude zjistit, zda se dá podvržením daného vstupu dosáhnout nepatřičného chování programu, které by mohl případný útočník zneužít ve svůj prospěch. Pokud by tomu tak bylo, pak bychom objevili vážnou zranitelnost programu, která by mohla znamenat ohrožení počítače uživatele. V takovém případě se pokusíme navrhnout, jak by se dal využitý mechanismus opravit, aby se zranitelnost v ideálním případě zcela eliminovala či aspoň co nejvíce omezila.

V poslední části této práce se pokusíme praktickou ukázkou demonstrovat rizika, která mohou nastat zneužitím nalezených zranitelností. Nejprve tedy navrhneme možné útoky a ty se pak pokusíme v testovacím prostředí vyzkoušet. V závěru této práce shrneme zjištěné poznatky a na jejich základě zhodnotíme bezpečnost programu ve své aktuální podobě. Také zde doplníme náš názor na rizika využívání tohoto softwaru, případně uvedeme doporučené nastavení během instalace, které potenciální zranitelnosti omezí či úplně eliminuje.

Kapitola 1

Teoretický postup analýzy

V úvodu této kapitoly se krátce vrátíme k volbě konkrétního programu, který budeme v rámci diplomové práce analyzovat. Shrneme zejména faktory, na jejichž základě jsme tuto volbu provedli. V následující části se zaměříme na to, jakým způsobem program komunikuje s okolím.

Během této komunikace totiž program může načítat data, která mohou významně ovlivnit jeho chování. Pokud by tedy dostatečně nekontroloval původ a obsah takto získaných dat, mohl by útočník tuto komunikaci zneužít k průniku do programu. V poslední části této kapitoly se zaměříme na jednotlivé programové nástroje, které lze k bezpečnostní analýze využít. Neklademe si za cíl informovat čtenáře o všech dostupných nástrojích, jen o jejich podmnožině, kterou budeme v rámci práce využívat.

1.1 Volba programu

Jak bylo řečeno již v úvodu, prvním provedeným krokem v rámci této diplomové práce byla volba zkoumaného programu. Naším hlavním kritériem bylo jeho relativně vysoké rozšíření mezi běžnými uživateli. Případná zranitelnost, kterou bychom objevili, by pak ohrožovala mnoho uživatelů. Tento fakt by mohl zaujmout i potenciálního útočníka, protože v případě úspěšné realizace útoku na tento program, by mohl snadno napáchat škody u více obětí. To samozřejmě pro útočníky zvyšuje lukrativnost hledání zranitelnosti v programu, a proto narůstá i pravděpodobnost zkoumání jeho bezpečnostních vlastností.

Tento požadavek rozšířenosti přispěl i k volbě operačního systému. Podle webového serveru *NetMarketShare* [14] využívá více než 90% uživatelů některou verzi operačního systému Windows od firmy Microsoft Corp., proto jsme se rozhodli dále zaměřit právě na programy určené pro tento operační systém. Abychom dokázali určit, které programy jsou relativně hojně používané,

rozhodli jsme se prozkoumat webové servery specializované na poskytování bezplatného softwaru. Mezi jejich běžné funkce totiž patří i žebříčky nejstahovanějších programů. Tyto žebříčky jsou zobrazovány nejspíše i z toho důvodu, aby usnadnili rozhodování těm uživatelům, kteří si nejsou jistí, jaký nástroj z dané kategorie si mají zvolit. Pro nás tato informace znamená přibližné měřítko o tom, jak je daný program oblíbený. Zde je nutné zdůraznit zejména to, že se jedná pouze o přibližnou informaci o oblíbenosti programu. Například fakt, že je program bezplatný, samozřejmě ještě nutně neznamená, že je rozšířenější než placená verze. Jako hrubé měřítko to však postačí. Serverem, který nám posloužil k výběru oblíbeného programu, je *stahuj.centrum.cz*. Tento server se svým působením zaměřuje na české uživatele a umožnil nám získat přehled o poměrně oblíbených programech v České republice.

Výsledkem našeho hledání je program Charlie, který se vyskytoval v listopadu roku 2014 mezi dvaceti nejstahovanějšími programy. Zároveň jsem ho již několik uplynulých let pozoroval ve svém okolí, a to i mezi svými příbuznými. Proto jsem se, po dohodě se svým vedoucím práce, rozhodl vybrat právě tento program. Jeho analýza mi totiž umožní posoudit, zda je program Charlie dostatečně bezpečný a zda nekompromituje bezpečnost počítačů mých příbuzných. Na základě takto získaných poznatků budu moci doporučit těmto lidem z mého blízkého okolí, jak program ideálně nastavit nebo, v případě, že se ukáže jako velmi nebezpečný, jim doporučit využití jiného nástroje s podobnou funkcionalitou. Zde bych rád zopakoval informaci z úvodu, a to že název programu Charlie, který v této práci využíváme, je pouze pseudonymem, a ve skutečnosti se tento program jmenuje odlišně. Tuto záměnu v názvu jsme provedli zcela úmyslně, protože tato práce bude veřejně dostupná a případný útočník by tedy mohl nalezené skutečnosti snadno zneužít ve svůj prospěch. Tomu chceme samozřejmě zamezit.

1.2 Komunikace aplikace s okolím

Potenciální útočník se většinou pokusí zneužít komunikace aplikace s okolním světem. Jak jsme zmiňovali již v úvodu této práce, tato komunikace může být realizována mnoha odlišnými způsoby. Každý z těchto způsobů využívá různé komunikační kanály. Mezi ně patří například registrové klíče, soubory, síťové sockety a mnohé další. V rámci bezpečnostní analýzy aplikace bychom se měli zaměřit na co nejvyšší procento (ideálně pak na všechny) z těchto způsobů. Každý z těchto komunikačních kanálů totiž může být útočníkem napaden a přenášená data mohou být pozměněna. Potřebujeme tedy prozkoumat zda, a případně jakým způsobem, aplikace ověřuje data přijatá z těchto komunikačních kanálů, aby skrze ně nemohl útočník program napadnout. V následujících podkapitolách se seznámíme s těmi druhy komunikačních kanálů, které budeme dále zkoumat podrobněji.

1.2.1 Registry operačního systému

Prvním z komunikačních kanálů, na který se v rámci bezpečnostní analýzy zaměříme, jsou registry operačního systému Windows (pro potřeby této diplomové práce je budeme dále nazývat pouze registry) [10]. Jedná se o databázi systému Windows, která obsahuje důležité informace o systémovém hardwaru, instalovaných programech a jejich nastavení a o profilech uživatelských účtů v daném počítači. Tato databáze je stromově uspořádána a obsahuje několik hlavních větví. Toto členění umožňuje logické uspořádání obsažených informací podle jejich účelu. Například do jedné z těchto větví můžeme uložit informace specifické pro daného uživatele a do jiné větve lze uložit informace závislé na konkrétní hardwarové konfiguraci počítače. Jednotlivé uzly této databáze nazýváme registrovými klíči. Registrový klíč může obsahovat jeden nebo více podklíčů, které jsou reprezentovány potomky konkrétního uzlu v databázi. Každý registrový klíč může kromě odkazů na své podklíče obsahovat i různé hodnoty. Tyto hodnoty reprezentují informace uložené v databázi.

Do této databáze registrových klíčů jsou ukládány i hodnoty, které ovlivňují chod počítače. Aby nebylo možné tyto hodnoty neautorizovaně upravovat, potřebujeme různým registrovým klíčům a jejich hodnotám přiřadit různé úrovně přístupových práv. Operační systém Windows k tomuto účelu využívá mechanismus *Access Control Lists*, zkráceně ACL [30]. Ten můžeme zjednodušeně popsat jako seznam přístupových oprávnění ke konkrétnímu objektu. Tímto objektem však nemusí být pouze registrový klíč, ale i soubory uložené v operačním systému, procesy, pojmenované roury a mnohé další. Pomocí těchto oprávnění je jednoznačně určeno, zda je uživateli přístup ke konkrétnímu objektu povolen, zamítnut či auditován. Auditován znamená, že lze k objektu přistoupit, ale tento přístup je zaznamenán a může být zpětně dohledán. V rámci této diplomové práce se budeme zabývat i kontrolou ACL u jednotlivých objektů.

1.2.2 Soubory v operačním systému

Dalším kanálem, na který se v rámci této diplomové práce zaměříme, jsou soubory. Program může za svého běhu pracovat s různými typy souborů, které obsahují informace ovlivňující jeho chování. Mezi tyto soubory řadíme různé konfigurační soubory, dokumenty, knihovny a mnohé další. Informace obsažené v souborech mají většinou předem danou strukturu, kterou program očekává. Zejména pokud by bylo jejich zabezpečení nedostatečné, tak je nutné vyzkoušet, jakým způsobem program tuto strukturu dat a obsažené informace před jejich dalším zpracováním kontroluje.

Mezi další soubory, které mohou v operačním systému Windows významně ovlivnit chod libovolného programu, patří skupina souborů nazývaná *PE sou-*

bory. Do této skupiny patří soubory s příponami SCR, CPL, DLL, EXE a další. V rámci této práce se budeme zabývat zejména DLL a EXE soubory. Ty jsou téměř totožné, liší se pouze v několika bitech a ve svém významu. Soubory s příponou EXE (dále jen EXE soubory) reprezentují soubory spustitelné v operačním systému Windows. Mohou obsahovat veškeré funkce potřebné ke svému běhu nebo mohou využívat dynamicky linkované knihovny (dále jen DLL soubory). Ty nelze přímo spustit, obsahují však jednu či více exportovaných funkcí, které lze v programu libovolně využít. Tento princip je vhodný zejména k využívání nástrojů třetích stran, například k vykreslování grafických objektů. Mnoho DLL souborů je poskytováno a udržováno operačním systémem. Ty budeme dále v této práci označovat jako systémové DLL soubory. Tento princip využívání dynamicky linkovaných knihoven umožňuje snadnou aktualizaci jednotlivých komponent programu — pokud některou z těchto knihoven aktualizujeme (například opravíme některou nalezenou zranitelnost), nemusíme měnit celý nainstalovaný program. Stačí vyměnit pouze tuto knihovnu. Tento přístup však může být zneužit útočníkem. V případě, že nejsou tyto DLL dostatečně chráněny, může podvrhnout vlastní, pozměněnou knihovnu, která umožní provedení útoku na program. Využití DLL souborů se může také stát nevýhodou, pokud je v některém z nich objevena a zveřejněna kritická chyba. Ta pak postihne všechny programy, které danou DLL využívají. Proto se v rámci této práce zaměříme i na jednotlivé verze DLL knihoven a prozkoumáme, zda program nevyužívá některou zranitelnou. PE soubory mohou obsahovat i staticky linkované knihovny. To znamená, že knihovna byla připojena během překladu a tvoří spolu s přeloženým zdrojovým kódem jeden výstupní soubor – DLL či EXE. Opět platí, že takto připojená knihovna může obsahovat nějakou již odhalenou zranitelnost. Proto se zaměříme i na kontrolu takto připojených knihoven a jejich verzí.

U mnoha zranitelností objevených v jiných aplikacích útočník podsunul programu svá vstupní data, která reprezentovala určitý vykonatelný kód. Operační systém Windows poskytuje různé mechanismy, kterými se snaží tento typ útoků co nejvíce zkomplikovat či úplně znemožnit. Mezi metody, které jsou dnes hojně využívané, patří i ASLR a DEP (Viz.1.2.2.1). Ty mohou být využívány jednotlivými PE soubory. Protože ztěžují útok na program a tím zvyšují jeho bezpečnost, zaměříme se v rámci bezpečnostní analýzy aplikace i na fakt, zda jeho PE soubory tyto mechanismy využívají.

V předchozích odstavcích jsme si představili některé ze souborů, které běžně program využívá ke svému běhu. Zdaleka se však nejedná o všechny - namátkou můžeme vyjmenovat například různé obrázky, certifikáty či dočasné soubory. V rámci této diplomové práce se pokusíme odhalit všechny soubory, které jsou během činnosti programu využívány. Tyto soubory jsou v operačním systému Windows opět chráněny pomocí ACL, které byly zmíněny dříve (1.2.1).

Proto se v tomto kroku musíme zaměřit i na kontrolu nastavení těchto ACL. Pokud bychom objevili některé potenciálně slabé (takto budeme v rámci této diplomové práce nazývat ACL, které umožňuje útočníkovi pozměnit soubor, registrový klíč či jiný objekt), tak bychom odhalili potenciální zranitelnost, kterou bychom měli v naší bezpečnostní analýze dále podrobněji prozkoumat.

1.2.2.1 ASLR a DEP

Nyní si stručně představíme ASLR a DEP. Jedná se o mechanismy operačního systému, jejichž cílem je snížit riziko zneužití programu a tím zvýšit jeho bezpečnost. Cílem ASLR neboli *Adress space layout randomization* je pseudonáhodně pozměnit uspořádání a tedy i adresy paměťových stránek, které jsou používané programem. Dochází k náhodnému pozměnění začátku zásobníku, haldy a kódového i datového segmentu [22]. Toto posunutí od očekávaného počátku se často nazývá *offset*. V závislosti na implementaci jsou tyto offsety určeny například při každém spuštění systému. Jiné implementace provedou tuto randomizaci během každého spouštění programu. Cílem použití ASLR je bránit případnému útočníkovi v odhadnutí adresy, na kterou program skočí a kterou bude chtít útočník pozměnit tak, aby došlo k vykonání jím zadaného kódu, takzvaného *exploitu*. Efektivita této operace je samozřejmě závislá například na velikosti virtuálního adresního prostoru, v rámci kterého může ASLR měnit adresy. Tento příznak je běžně podporován téměř na všech moderních operačních systémech, nicméně stále jsou rozšířené PE soubory, kterého ho nevyužívají a ulehčují tak útočníkovi práci.

Druhým zmiňovaným mechanismem je *Data execution prevention* (DEP). Jedná se o možnost označit data jako nespustitelná. Data jsou, stejně jako kód, načítána do paměti využitím stránek. Tyto stránky mají svá přístupová práva, která určují, zda mohou být čteny, zapisovány či spuštěny. Pokud využíváme DEP, tak jsou stránky v paměti obsahující data programu označena jako nespustitelná. Tím je efektivně zamezeno vykonání kódu, který se podařilo útočníkovi skrýt v datové části programu. Stejně jako u ASLR, i v tomto případě platí, že DEP je podporován všemi moderními operačními systémy. Stále však existuje poměrně mnoho PE souborů, které ho nevyužívají, a proto se na kontrolu jeho využití také zaměříme.

1.2.3 Komunikace po síti

Protože jednotlivé programy často získávají informace potřebné ke svému běhu z vnějšího světa pomocí internetu, bude posledním komunikačním kanálem, na který se v této diplomové práci zaměříme, právě komunikace programu po síti pomocí protokolu TCP/IP. Mezi data, která jsou tímto způsobem často získávána, patří například automatické aktualizace programu, vzhled aplikace, tlačítek a mnohé další. Zejména tento komunikační kanál by měl být velmi

dobře zabezpečen a data získaná jeho prostřednictvím by měla být velmi důkladně ověřována. Jedná se o důležitý krok, protože je známo velké množství útoků, které umožňují data během komunikace po síti pozměnit či zaměnit jejich zdroj.

1.2.4 Shrnutí

V předchozích podsekcích jsme se seznámili se třemi základními způsoby komunikace zkoumaného programu: komunikace s registry, souborovým systémem a po síti. U všech těchto zmíněných zdrojů dat se musíme ze všeho nejdříve zaměřit na kontrolu jejich zabezpečení. Tím myslíme zejména to, zda mohou být data snadno podvržena útočníkem, což by mohlo vést k odlišnému chování programu, než jaké jeho vývojáři původně zamýšleli. Toto podvržení dat může nastat zejména v případě slabých ACL u registrů a souborů či použitím nezabezpečených komunikačních protokolů u komunikace po síti. Zejména v takovém případě je nutné ověřit, zda a jak ověřuje program původ a obsah u získaných dat.

1.3 Využité nástroje

V předchozí sekci jsme se seznámili s jednotlivými komunikačními kanály programu, na které se v dále zaměříme. Abychom mohli tuto analýzu snáze uskutečnit, tak je vhodné využívat některé specializované nástroje. V následujících podsekcích se zaměříme na ty, které byly v rámci této diplomové práce využívány. Téměř pro každou zde uvedenou akci se však dá na internetu nalézt mnoho různých nástrojů. Námi zmiňované programy mají společné zejména to, že jsou volně dostupné.

1.3.1 Práce s registry

Jedním z komunikačních kanálů, na který jsme se rozhodli během bezpečnostní analýzy programu Charlie zaměřit je komunikace tohoto programu s registrovými klíči. K usnadnění tohoto kroku potřebujeme naleznout takové nástroje, které dovedou jednotlivé operace (vytvoření, otevření, změna, smazání, ...) s registrovými klíči monitorovat. Tím získáme přehled o registrových klíčích, které jsou programem využívány. Abychom se mohli později k takto nalezeným registrům vrátit, měl by nástroj umožňovat jednoduché uložení zobrazených výsledků.

1.3.1.1 Regshot

Prvním nástrojem, na který se zaměříme je program Regshot [17]. Jedná se o aplikaci s otevřeným zdrojovým kódem, uvolněnou pod licencí LGPL. Regshot je úzce specializovaný nástroj, který vytvoří záznam stavu registrů

v určitém časovém okamžiku a umožní ho později porovnat s druhým, podobně vytvořeným záznamem. Výstupem jsou pouze registrové klíče, ve kterých proběhla nějaká změna: byly vytvořeny, smazány či upraveny. Pokud tedy uložíme stav systému před a po analyzované operaci, například instalaci nového softwaru, lehko zjistíme, které registrové klíče byly instalací doplněny. Na ně se pak můžeme zaměřit v další analýze. Dá se totiž předpokládat, že takto uložené klíče budou v určitý okamžik použity jako vstup do našeho programu.

1.3.1.2 Process Monitor

Process Monitor [21] od firmy Sysinternals zaznamenává a v reálném čase zobrazuje velkou škálu událostí, které jednotlivé procesy a jejich vlákna během svého běhu provádějí. Protože je v operačním systému v jednom časovém okamžiku spuštěno mnoho procesů, které vykonávají různé činnosti, je většinou maximálně během pár minut nashromážděno několik milionů řádků dat. Po provedení požadované operace je tedy vhodné sběr dat zastavit a nástroj využít již jen k analýze takto nashromážděných dat. Samozřejmostí je možnost uložit takto získaná data do výstupního souboru ve svém nativním formátu či ve formátu CSV.

Mezi shromažďované údaje jsou zahrnuty operace se souborovým systémem, jako je vytvoření, otevření či zápis souboru, operace s registrovými klíči či základní síťové události – například, že byl odeslán IP packet určité velikosti na danou IP adresu pomocí protokolu TCP. Jak již bylo zmíněno, množství takto nashromážděných dat je i během krátkého časového okamžiku poměrně rozsáhlé. Abychom se v datech dále lépe orientovali, tak je nutné je dále omezovat a filtrovat. To lze opět provádět přímo z tohoto programu. Není problém vyloučit procesy, které nejsou pro naše další zkoumání zajímavé nebo se můžeme snadno omezit jen na určité operace – například na vytváření souborů. Také lze zobrazovat události pouze z určitého časového úseku. Tyto filtrovací operace jsou poměrně důležité, protože významně usnadňují orientaci v nashromážděných datech. Užitečnou vlastností programu je to, že takto filtrovaná data můžeme opět uložit. Tím se do výstupního souboru zapíšou pouze zajímavé informace a často se tak podstatně omezí jeho velikost.

Firma Sysinternals byla v roce 2006 akvizována společností Microsoft, což naznačuje nynější provázanost jednotlivých aplikací s operačním systémem Windows. Všechny nástroje, v tomto případě Process Monitor, jsou neustále vyvíjeny a odlaďovány tak, aby umožnily perfektní kooperaci s operačním systémem a sběr veškerých relevantních dat.

1.3.2 Soubory

V této podsekci se seznámíme s nástroji, které lze využít k bezpečnostní analýze souborů využívaných testovanou aplikací. U tohoto komunikačního kanálu je nutné kontrolovat mnoho různých detailů, proto není možné vybrat pouze jediný nástroj, pomocí kterého bychom provedli kompletní analýzu. Se soubory úzce souvisí nastavení jejich přístupových práv, a proto si představíme nástroj, který je dokáže přehledně zobrazit. K odhalení staticky linkovaných knihoven se zaměříme na řetězce v binárním souboru, které jsou složené z běžných znaků. V nich mohou být obsaženy například informace o použitých knihovnách a jejich verzích. Dále se zaměříme na nástroje, které umožní snadno prozkoumat strukturu PE souborů. V závěru této části se zaměříme na ladící nástroje, které umožňují monitorovat jednotlivé PE soubory při běhu testovaného programu.

1.3.2.1 AccessEnum

AccessEnum [1] je dalším nástrojem původem od firmy Sysinternals, který je obsažen v jejím balíčku souborů: *Sysintrnals suite*. Slouží zejména ke kontrole nastavení jednotlivých přístupových oprávnění k danému adresáři a jeho podadresářům či k registrovým klíčům. Uživatel vybere registr či adresář, který chce otestovat a program automaticky prověří i všechny jeho podadresáře (případně podklíče). Ve výsledném výpisu jsou zobrazena přístupová práva k námi vybranému adresáři (registrovému klíči). Dále si můžeme vybrat, zda chceme zobrazit pouze soubory s méně omezenými právy než má tento rodič nebo zdali chceme zobrazit informace o všech souborech, které mají odlišná práva.

1.3.2.2 Strings

Strings [20] je úzce specializovaný nástroj od firmy Microsoft (dříve SysInternals), který je spustitelný v příkazovém řádku. Pomocí prvního parametru určíme vstupní soubor a Strings vypíše na standardní výstup pouze řetězce složené z běžných znaků. Tyto řetězce jsou identifikovány v rámci ANSI i UNICODE kódování, případně lze při spuštění zvolit jednu z variant. K zobrazení výsledků je používán standardní výstup příkazového řádku, který však lze snadno přesměrovat do souboru, a tak výsledky uchovat pro pozdější zkoumání. Takto nalezené informace se mohou hodit při identifikaci staticky linkovaných knihoven. Ty v sobě totiž často musí mít obsaženy copyrighty s informací o použité verzi knihovny, což je velmi užitečná informace při hledání dříve objevených zranitelností.

1.3.2.3 Binwalk

Binwalk [3] je volně dostupný nástroj, autorem umístěný na serveru GitHub. Tento nástroj prozkoumá celý poskytnutý soubor a zobrazí, zda v sobě obsahuje vnořené další soubory. Těmi mohou být komprimovaná data, copyrighty, XML dokumenty a mnohé další. Nástroj umožňuje použít i další funkce, například provést analýzu entropie a získané výsledky zobrazit v přehledném grafu. Je určen k využití v linuxovém operačním systému, dokáže však prozkoumat i PE soubory operačního systému Windows.

1.3.2.4 CFF Explorer

Jako další uvedeme program CFF Explorer [15]. Jedná se o volně dostupný nástroj, který plně podporuje analýzu PE struktury pro 32bitové i 64bitové Windows. Jeho výhodou je možnost nejen přehledně zobrazovat, ale i snadno upravit jednotlivé položky struktury konkrétního PE souboru. Tento nástroj v sobě obsahuje i další nástroje, které provedení analýzy usnadňují. Jedním z nich je hexadeximální textový editor, který umožňuje nahlédnout na data v jejich surové podobě. Pomocí dalšího z těchto vestavěných nástrojů můžeme snadno prozkoumat, které zdroje (ikony, tlačítka, ...) zkoumaný program využívá.

CFF Explorer zobrazuje i veškeré hlavičky, které binární soubor obsahuje. Mezi ně se řadí DOS hlavičky, hlavičky sekcí a hlavičky NT. Také přehledně zobrazuje, které funkce jsou do programu importovány z jiných knihoven, a které jsou naopak programem exportovány (mohou být využívány dalšími programy a knihovnami). Jak již bylo zmíněno, mezi velmi užitečnou vlastnost tohoto nástroje patří možnost úpravy binárního souboru.

1.3.2.5 Dumpbin

K prozkoumání struktury binárních souborů můžeme použí i nástroj Dumpbin [7]. Ten je určen pro příkazový řádek a je součástí balíku vývojových nástrojů Microsoft Visual Studio. Umožňuje vypsat informace o jednotlivých hlavičkách přímo do příkazového řádku a získat tak podobné informace jako při využití dříve představeného nástroje CFF Explorer.

1.3.2.6 Dependency Walker

Posledním nástrojem, který nám pomůže se zorientovat v programu a usnadní případné ladění, je Dependency Walker [12]. Jedná se opět o volně dostupný nástroj, který nejprve prozkoumá EXE či DLL soubor a poté sestaví hierarchický strom všech závislých knihoven a jiných Windows modulů (mezi ně patří například EXE, DLL, OCX, SYS a další typy souborů). Pro každou

takto nalezenou knihovnu zobrazí všechny funkce, které exportuje a také které z nich jsou volány ostatními moduly.

1.3.2.7 Dynamická analýza programu

Dosud představené nástroje nám umožnili statickou analýzu programu. Tu můžeme provádět bez spuštění programu. Nyní se zaměříme na nástroje, které naopak umožňují zkoumat vlastnosti a chování spuštěného programu. Tomuto přístupu se říká dynamická analýza. Hlavním přínosem je, že nám umožňuje pochopit, jak program reaguje na specifické události. K tomuto typu analýzy se nejčastěji využívají ladící nástroje. Ty umožňují podrobné sledování běžícího programu i jeho pozastavení v požadovaném okamžiku. Těchto nástrojů je velké množství a záleží pouze na preferencích každého z nás, který z nich využijeme. Užitečnou vlastností této skupiny programů je možnost připojení se k běžícímu procesu. Tím je nám umožněno monitorovat, jak se zkoumaný program chová „uvnitř“. Ve chvíli, kdy se k němu připojíme, dojde k jeho pozastavení a lze snadno prohlédnout jeho aktuální stav. Do tohoto stavu zahrnujeme například informace o tom, na které virtuální adresy byly připojeny jednotlivé DLL knihovny. V následujících odstavcích si představíme některé z této skupiny nástrojů.

WinDbg Často používaným ladícím nástrojem v operačním systému Windows je WinDbg [11]. Jedná se o oficiální ladící nástroj od firmy Microsoft Corp., který je často využíván i vývojáři této firmy k ladění jejich produktů. V našem systému ho nalezneme, například pokud používáme Microsoft Visual Studio k vývoji aplikací. Pokud tomu tak není, je volně dostupný na webu společnosti Microsoft. Výhodou tohoto programu je, že dovede ladit programy sestavené pro 64bitový operační systém stejně tak jako pro 32bitový. Tento nástroj lze ovládat pomocí myši a voleb grafického rozhraní nebo můžeme využít vestavěný příkazový řádek spolu s příkazy, které jsou pro WinDbg specifické.

Ollydbg Dalším ladícím nástrojem, který je při zpětném inženýrství často využíván je Ollydbg [28]. Jedná se o volně dostupný nástroj, autor však požaduje provedení bezplatné registrace. Ollydbg zatím umožňuje ladit jen 32bitové verze programů, ty jsou však stále hojně využívané. Jeho funkce lze snadno rozšířit pomocí velkého množství volně dostupných skriptů. Užitečnou vlastností je i možnost upravovat kód běžícího programu za jeho běhu a takto upravený program uložit.

IDA Pro Free Poslední nástroj, který si v této diplomové práci představíme, je IDA Pro Free [4] od firmy Hex-Rays. Program je vyvíjen přímo pro zpětné inženýry k usnadnění jejich práce. IDA Pro není jen ladící nástroj, umožňuje i pokročilou statickou analýzu PE souboru. Ten je nejprve načten

a poté se provede jeho analýza. IDA Pro Free postupně prozkoumá celý soubor a jednotlivé instrukce strojového kódu rozdělí do logických bloků podle toho, jak jsou vzájemně volány. Tyto bloky pak zobrazí pomocí přehledného vývojového diagramu, což usnadňuje analýzu jednotlivých funkcí. Do těchto vývojových diagramů můžeme umístit zarážky, na kterých bude program během svého ladění pozastaven.

Tato verze programu byla na trh uvolněna bezplatně, jedná se však o starou verzi (5.0), na které si zpětní inženýři mohou zdarma vyzkoušet základní možnosti, které poskytuje. Od jejího původního vydání však bylo uveřejněno několik nových verzí (v dubnu 2015 je aktuální verzí 6.8), které obsahují mnoho nových funkcí. Pro časté používání tedy musíme doporučit pořídit si komerční a tedy i placenou verzi.

1.3.3 Komunikace programu po síti

V této podsekci se seznámíme s nástroji, které umožňují analýzu síťové komunikace. U tohoto komunikačního kanálu rozhoduje o bezpečnosti mnoho různých detailů, které budeme muset důkladně prozkoumat. Mezi tyto detaily řadíme adresy komunikujících stran, použité síťové protokoly a podobně. Použitý nástroj by měl dovést co nejvíce z těchto informací zaznamenat a umožnit nám tak jejich další průzkum. Užitečnou vlastností, která zkoumání ulehčí je filtrace a uložení zaznamenaných výsledků.

1.3.3.1 TCPView a TCPvcon

První nástroj ulehčující analýzu síťové kumunikace, na který se zaměříme, je TCPView [19] z balíčku programů firmy Microsoft (dříve Sysinternals). Tento nástroj umožňuje detailní zobrazení veškerých TCP a UDP spojení, a to včetně lokálních a vzdálených adres, stavu TCP spojení či informací o tom, kolik bylo přeneseno během spojení dat. Samozřejmostí je přirazení takto zobrazených aktivních spojení ke svým procesům. Ty můžeme rozlišovat pomocí názvu procesu a pomocí PID (*Process identifier*, což je jednoznačný identifikátor procesu). Tento program má i svou verzi pro příkazový řádek, která se jmenuje TCPVcon a poskytuje naprosto shodnou funkcionalitu. Program započne zobrazovat veškerá aktivní spojení ve chvíli svého spuštění. Jedná se o pohled na jednotlivá otevřená spojení v reálném čase. Pokud je některé ze zobrazených spojení ukončeno, tak ho ze zobrazovaného seznamu vyloučí. Samozřejmostí je možnost pozastavit aktualizace tohoto seznamu, což nám umožní detailněji analyzovat některé z těch spojení, které trvají relativně krátký časový okamžik. Také je možné uložit aktuální seznam či vybranou podmnožinu do textového souboru, který můžeme analyzovat později.

1.3.3.2 Netstat

Netstat [8] je program od společnosti Microsoft, který pracuje pouze v příkazovém řádku a v operačním systému je předinstalovaný. Umožňuje nám zobrazit podobné informace jako dříve představený TCPView. Opět tedy můžeme snadno prohlédnout veškeré aktivní TCP a UDP spojení. Tento seznam je bez použití parametrů zobrazen pouze jednorázově a jedná se o aktuální stav spojení. Pokud nás zajímá, kdo využívá konkrétní spojení, tak můžeme pomocí parametru při spuštění program požádat o zobrazení PID procesu. Netstat je určen k detailnější analýze stavu počítačových sítí a dokáže tedy zobrazit mnohem více informací. Mezi tyto informace můžeme například zařadit údaje ze směrovacích tabulek daného zařízení, případně statistiky jednotlivých síťových protokolů. Tyto informace však pro nás nejsou většinou podstatné, protože ve většině případů by běžné programy neměly zmíněné informace a údaje nijak ovlivňovat.

1.3.3.3 Wireshark

Wireshark [27] je nástroj společnosti Wireshark Foundation určený k analýze komunikace po síti. Umožňuje zaznamenávat veškerou síťovou komunikaci na určeném síťovém rozhraní. Jsou tedy odchyceny a zaznamenány veškeré pakety, které byly přes toto rozhraní odeslané. Umožňuje zobrazit různé detaily o probíhající komunikaci a pokud se jedná o nezabezpečenou komunikaci, pak můžeme snadno zobrazit i přenášená data. To může být využito například pro odhalení zranitelnosti, kdy testovaná aplikace odesílá přihlašovací údaje v nešifrované podobě. Samozřejmostí je možnost filtrace konkrétního spojení či komunikace s konkrétní IP adresou. To usnadňuje práci zejména v případě velkého vytížení síťového rozhraní, přes které komunikace probíhá.

1.3.3.4 Shrnutí

V této podsekci jsme se seznámili s různými nástroji, které lze využít k analýze jednotlivých komunikačních kanálů. K získání informací o práci s registrovými klíči jsme se rozhodli využívat nástroj Process Monitor. Oproti nástroji Regshot umožňuje monitorovat i operace, které zkoumaný program provede v souborovém systému, a získat tak informace, které můžeme využít právě během analýzy souborů.

Statickou analýzu jednotlivých PE souborů provedeme pomocí nástroje CFF Explorer, který z uvedených nástrojů poskytuje nejvíce informací. K dynamické analýze chování programu využijeme nástroje WinDbg a IDA Pro Free. WinDbg umožňuje, narozdíl od zmíněného OllyDgb, analyzovat i 64bitovou verzi programu a IDA Pro Free poskytuje přehledné vývojové diagramy zkoumaných funkcí.

K analýze síťové komunikace použijeme nástroj Wireshark, který poskytuje v přehledné formě podrobné informace o celé komunikaci. Nástroje TCPView a Netstat nedovedou zobrazit obsah přenášených dat, pouze informace s kým program komunikuje.

1.3.4 Závěr teoretické analýzy

V této části diplomové práce jsme nejprve provedli volbu programu, který budeme dále zkoumat. Hlavním kritériem pro tuto volbu byla jeho rozšířenost mezi uživateli. To jsme hodnotili pomocí statistik na serveru stahuj.centrum.cz. Protože tato práce bude veřejně dostupná v elektronickém archivu, nechceme poskytovat návod ke zneužití možných zranitelností, budeme zvolený program nazývat pouze pseudonymem Charlie a v přiložených datech zakryjeme ([redacted]) či zobecníme veškeré informace, které by mohly vést k jeho identifikaci.

Dále jsme se zaměřili na komunikační kanály do programu, kterými jsou registrové klíče, soubory uložené na disku počítače a síťová komunikace programu. Všechny tyto kanály mohou být potenciálně zneužity útočníkem. Toto zneužití může znamenat pozměnění přenášených dat útočníkem, a proto bude nutné tuto situaci dále analyzovat. Nakonec jsme shrnuli nástroje, které budeme používat pro analýzu těchto komunikačních kanálů.

V další kapitole se budeme zabývat analýzou instalace programu Charlie. Pomocí zvolených nástrojů se zaměříme na kontrolu využívání zmíněných komunikačních kanálů.

Kapitola 2

Analýza instalace programu

V předchozí části této diplomové práce jsme zvolili program Charlie, kterému se budeme dále věnovat. Také jsme, jako potenciální slabá místa libovolného programu, identifikovali komunikační kanály. Mezi ně patří registrové klíče, soubory uložené na disku a síťová komunikace. Tyto komunikační kanály budeme zkoumat i u programu Charlie. Protože používání programu začíná již jeho instalací, rozhodli jsme se rozdělit bezpečnostní analýzu programu Charlie do dvou kroků. V této kapitole se zaměříme na proces instalace programu Charlie a v kapitole následující na program samotný.

V této kapitole postupně provedeme bezpečnostní analýzu jednotlivých komunikačních kanálů. K tomu využijeme nástroje, se kterými jsme se seznámili v minulé kapitole. Zaměříme se na analýzu souborů, které instalační proces zapisuje na disk. K tomu použijeme nástroj Process Monitor (1.3.1.2). Tento nástroj zaznamenává mnoho operací, které jednotlivé procesy v operačním systému Windows provádí, a proto ho využijeme i k analýze registrových klíčů, do kterých instalační proces zapisuje. Pomocí nástroje CFF Explorer (1.3.2.4) prozkoumáme i stažený instalační soubor a ověříme, které knihovny využívá ke svému běhu. Posledním komunikačním kanálem, na který se v analýze instalačního procesu zaměříme, je síťová komunikace. K jejímu prozkoumání využijeme nástroj Wireshark (1.3.3.3).

2.1 Konfigurace instalačního programu

Mnoho běžných uživatelů provádí instalaci programu rychlým potvrzením všech možností, nejspíše proto, aby je instalace samotná příliš nezdržovala a brzy si mohli užít služeb vytouženého programu. To přenáší velkou míru zodpovědnosti na vývojáře tohoto programu. Musí totiž přednastavit výchozí volby instalačního programu tak, aby uživatele neohrozil potenciálně nebezpečným nastavením. K tomu se využívá bezpečnostního principu *Secure by default* [5], kdy instalační proces využívá takové výchozí nastavení, které je

nejbezpečnější pro uživatele. Abychom mohli tento princip ověřit, rozhodli jsme se ponechat nastavení instalačního procesu na jeho výchozích hodnotách.

Instalační program je podepsán digitálním certifikátem, který je potvrzený firmou VeriSign, Inc., jejíž kořenový certifikát je ve Windows zařazen mezi důvěryhodné certifikační autority. To znamená, že po spuštění instalačního procesu, označí operační systém Windows vydavatele softwaru jako důvěryhodného (v dialogovém okně *Řízení uživatelských účtů*). Velikost instalátoru je necelý 1 MiB a to navzdory tomu, že program Charlie obsahuje mnoho funkcí. To naznačuje, že instalátor bude pravděpodobně získávat další data přes internet. Instalace podle předpokladů vyžaduje oprávnění systémového administrátora.

Instalační proces se nejprve uživatele zeptá, zda chce program Charlie asociovat s podporovanými soubory. Tato volba je nastavena jako výchozí. V dalším kroku je uživatel dotázán, zda chce použít doporučenou plnou instalaci, která je nastavena jako výchozí, nebo zda použije volitelnou instalaci, ve které může blíže specifikovat které kroky instátor provede. V plné instalaci je zahrnuta instalace samotného programu Charlie, doplnění panelu do internetového prohlížeče a nastavení nové domovské stránky prohlížeče.

Zbytek instalace proběhl zcela automaticky bez interakce s uživatelem. Kompletní instalační proces jsme monitorovali pomocí dříve představených nástrojů Process Monitor (1.3.1.2), Wireshark (1.3.3.3), TCPView (1.3.3.1) a Regshot (1.3.1.1). Instalační program nainstaloval program Charlie i panel do prohlížeče Internet Explorer (dále pouze panel). Tento panel může ovlivnit bezpečnost užitele, a proto se v dalších částech této diplomové práce budeme zabývat analýzou programu Charlie i tohoto panelu.

2.2 Zápis souborů

Pomocí nástroje Process Monitor jsme získali podrobný přehled o adresářích souborového systému, do kterých instalační proces zapisoval instalovaná data:

- C:\Program Files (x86)\[redacted]\Charlie
- C:\Program Files (x86)\[redacted]\Panel
- %USERPROFILE%\AppData\Roaming\Charlie
- %USERPROFILE%\AppData\LocalLow\Panel
- %USERPROFILE%\AppData\Local\Panel
- C:\ProgramData\Panel\IE

Dále instalační proces využíval adresáře k uložení dočasných souborů či přistupoval k systémovým knihovnám. Dočasné soubory však nebudou programem Charlie dále využívány, a proto nebyly v předcházejícím výpise zobrazeny. Můžeme si povšimnout, že rozložení souborů a adresářů je rozděleno na ty, které patří samotnému programu Charlie a na soubory panelu. To uživateli umožňuje v případě potřeby snadno odebrat pouze jeden z těchto nástrojů a druhý v operačním systému ponechat.

Adresář *C:\Program Files (x86)* slouží primárně pro ukládání souborů aplikací nainstalovaných v operačním systému. Do tohoto adresáře mohou zapisovat pouze uživatelé a procesy s oprávněním systémového administrátora. Čtení souborů je povoleno všem místním uživatelům. Naopak adresář *%USERPROFILE%\AppData* má přístupová práva nastavena mírněji a umožňuje zápis i uživateli, kterému patří. Z tohoto důvodu je většinou využíván pro uložení konfiguračních souborů takových komponent programu, které závisí na osobním nastavení každého uživatele. Posledním adresářem, který byl instalátorem využit k uložení dat je *C:\ProgramData*. Přístupová práva tohoto adresáře umožňují libovolnému uživateli zobrazit a spustit obsažené soubory. K jejich úpravě musí mít uživatel či proces oprávnění systémového administrátora. Do tohoto adresáře byl uložen EXE soubor umožňující odinstalování panelu.

2.3 Zápis registrů

Změny v registrech operačního systému jsme monitorovali pomocí programů Process Monitor (1.3.1.2) a Regshot (1.3.1.1). Pozorovaný instalační program nejvíce využívá uživatelskou část databáze registrů, tedy *HKEY Current User* (dále jen HKCU). Jak i název této části databáze registrů napovídá, obsahuje registrové klíče specifické pro konkrétního uživatele. Tomu odpovídají i nastavená přístupová práva, kdy uživatel sám může hodnoty ovlivnit a změnit. Dále se registrové klíče zapisují do *HKEY Local Machine* (HKLM) a *HKEY Current Root* (HKCR), do kterých instalační proces uloží zejména informace týkající se registrovaných COM komponent. Poslední používanou větví v databázi registrů je *HKLM\SOFTWARE\Wow6432Node*. Ta je v registrech umístěna, pokud používáme 64bitový operační systém, abychom mohli používat i 32bitové verze programů. Níže si můžeme prohlédnout registrové větve, do kterých bylo během instalace zapisováno:

- HKLM\SOFTWARE\Classes\CLSID\
- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion...
  ...\Explorer\[redacted]\
- HKLM\SOFTWARE\Wow6432Node\[redacted]
- HKCR\CLSID

- HKCU\Software\AppDataLow\Software\[redacted]
- HKCU\Software\AppDataLow\Software\[redacted]
- HKCU\Software\[redacted]
- HKCU\Software\Classes
- HKCU\Software\[redacted]
- HKCU\Software\[redacted]
- HKCU\Software\[redacted]\AppPaths\[redacted]

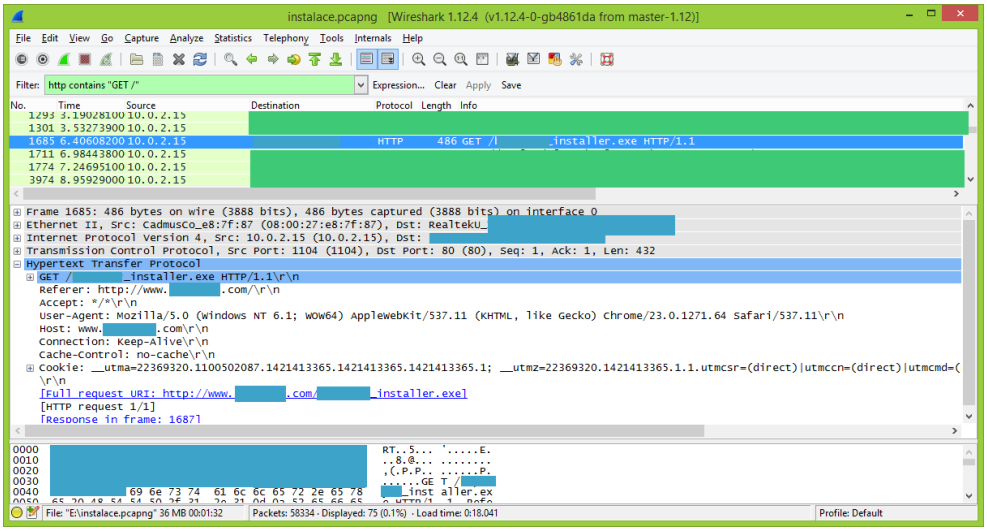
2.4 Síťová komunikace

Abychom mohli potvrdit dříve učiněný předpoklad, a to že instalační proces získává data potřebná pro svůj běh z internetu, sledovali jsme během celé instalace i síťovou komunikaci instalačního procesu. Protože Process Monitor (1.3.1.2) neposkytuje pro tento typ komunikace dostatečně detailní informace, použili jsme nástroje TCPView (1.3.3.1) a Wireshark (1.3.3.3), které poskytují informace o přenesených paketech v dostatečném rozsahu a tím umožňují jejich bližší analýzu. Předpoklad síťové komunikace instalačního procesu se potvrdil, když jsme pomocí Wiresharku odchytili rozsáhlou komunikaci mezi tímto procesem a několika servery. Doménová jména těchto serverů zde nebudeme uvádět, abychom neposkytli vodítko k odhalení skutečného názvu programu Charlie.

Z průběhu celé instalace se ukázala být nejpodstatnější ta část, která je zvýrazněná na obrázku 2.1. Jak si můžeme povšimnout, instalátor stahuje z domény *www.*[redacted]*.com* spustitelný soubor *charlie_installer.exe*. Podle názvu tohoto souboru můžeme učinit oprávněný předpoklad, že se jedná o instalaci vlastního programu Charlie. Ekvivalentní řádek byl nalezen i pro instalaci panelu internetového prohlížeče. Pozorovaný instalační program je tedy spíše prostředník, který má za úkol nabídnout doplněk hlavního programu ve formě volitelného panelu prohlížeče a umožňuje zamítnout asociaci programu Charlie s typy souborů, které dokáže zpracovat. V dalším kroku instalátor stáhne jednotlivé dílčí instalační programy a spustí je. K získání těchto dílčích instalátorů se využívá nezabezpečený protokol, a proto by měl instalační proces takto získaná data důkladně prověřit.

2.5 Struktura PE souborů

Pro úplnost jsme se zaměřili i na binární strukturu jednotlivých instalačních programů – tedy hlavního instalačního programu i dílčích, které jsou stažené



Obrázek 2.1: Instalace - získávání dílčího instalátoru

v průběhu jeho činnosti. Pomocí nástroje Strings (1.3.2.2) jsme se získali seznam použitých staticky linkovaných knihoven. Využitím nástroje CFF Explorer (1.3.2.4) jsme odhalili, které dynamicky linkované knihovny, instalační procesy využívají. Takto získané knihovny jsme prozkoumali s využitím online databáze známých zranitelností [13]. Tímto způsobem jsme neodhalili žádnou známou zranitelnost. Nástroj CFF Explorer jsme využili i ke kontrole PE hlaviček souborů. Zjistili jsme, že žádný z instalačních programů nebyl přeložen s využitím ASLR či DEP.

2.6 Pozorované vlastnosti

Jak již bylo zmíněno, instalační program se snaží pracovat co nejvíce automaticky a instalaci uživateli co nejvíce usnadnit tím, že téměř veškeré volby provede za něj. Poté stáhne dílčí instalační procesy a postupně provede i jejich spuštění a instalaci. Stažením a spuštěním jsme ověřili, že jednotlivé instalační soubory nainstalují požadované komponenty.

Hlavní instalační program je podepsán platným certifikátem, díky kterému ho operační systém označuje za důvěryhodný proces. Protože však spouští další instalační soubory, rozhodli jsme se zkontrolovat i ty. Odkazy k jejich získání jsme získali během odposlechu síťové komunikace programem Wireshark (1.3.3.3). Oba tyto soubory (instalační proces samotného programu Charlie i jeho panelu do prohlížeče) jsou opět podepsány platnými certifikáty, což umožňuje jejich případné ověření pomocí kontroly certifikátu. Dalším společným znakem je fakt, že pokud je spustíme sami, tak oba nabízejí další

konfigurační volby. Jedná se o možnosti, na které jsme byli zvyklí u dříve vydávaných programů: kam se má program nainstalovat, zda se má vytvořit zástupce programu na ploše uživatele a podobně. Hlavní instalační program tedy nejspíše jednotlivé volby programu rovnou potvrzuje. V dalším kroku bylo nutné ověřit, zda a jak hlavní instalační program prověřuje stažená data. Během bližšího průzkumu pomocí nástrojů WinDbg (1.3.2.7) a IDA Pro Free (1.3.2.7) se nám toto ověřování nepodařilo naleznout.

2.7 Závěry z pozorování instalace

V této kapitole jsme se zaměřili na kontrolu instalačního procesu aplikace Charlie. Prozkoumali jsme výchozí nastavení instalačního procesu. Zjistili jsme, že spolu s vlastním programem Charlie je instalován i panel do prohlížeče Internet Explorer. Získali jsme seznam adresářů a registrových klíčů, do kterých bylo během instalace zapisováno. Tyto informace použijeme v následující kapitole, kde nám usnadní analýzu programu Charlie i jeho panelu.

Dále jsme se zaměřili na kontrolu průběhu instalačního procesu. Podařilo se nám zjistit, že instalační program získaný z originálních webových stránek projektu slouží zejména k volbě jednotlivých komponent, které si uživatel přeje nainstalovat. Tyto dílčí komponenty pak stahuje z webu a automaticky instaluje do počítače. Navzdory tomu, že všechny spustitelné soubory jsou podepsané platnými certifikáty a tomu, že pomocná data jsou z webu stahována využitím nezabezpečených síťových protokolů, nejspíše nepoužívá hlavní instalátor žádnou kontrolu takto získaných dat. V tabulce 2.1 si můžeme prohlédnout odhalené hrozby.

Tabulka 2.1: Objevené hrozby - instalace procesu Charlie

Hrozba	**Riziko**
Veškerá síťová komunikace probíhá pomocí protokolu HTTP	Střední
Dílčí instalace nevyužívají ASLR ani DEP	Malé
Instalační program neověřuje získaná data	Velmi vysoké

V následující kapitole se zaměříme na bezpečnostní analýzu samotné aplikace Charlie a jejího panelu. Pro vyšší přehlednost budeme jednotlivé kroky analýzy provádět pro oba tyto programy nezávisle. Postupně se zaměříme na jednotlivé komunikační kanály, kterými získávají svá data. Mezi tyto kanály zahrnujeme práci se soubory uloženými na disku zařízení, registrovou databází a na síťovou komunikaci programů.

Kapitola 3

Analýza programu Charlie a jeho panelu

V této kapitole se zaměříme na analýzu programu Charlie a panelu internetového prohlížeče, který byl nainstalován spolu s ním. Protože se jedná o dva různé produkty, provedeme jednotlivé kroky zkoumání nezávisle. Pomocí nástrojů, které jsme vybrali v první kapitole, prozkoumáme jednotlivé komunikační kanály, které programy využívají.

Prvním komunikačním kanálem, který prozkoumáme jsou soubory. Nejprve se zaměříme na kontrolu PE souborů, které obě dílčí aplikace ke svému běhu využívají. Zkoumáním těchto souborů získáme seznam knihoven (staticky i dynamicky linkovaných), které používají. Tento seznam porovnáme s online databází známých zranitelností. Ve spojení s touto kontrolou je užitečné provádět i kontrolu PE hlaviček těchto souborů. Tím zjistíme, zda využívají mechanismy operačního systému (ASLR a DEP), které znesnadňují některé metody zneužití zranitelností. Během pozorování instalace jsme získali seznam souborů a registrových klíčů, které jsou do počítače nainstalovány. Tento seznam nyní prozkoumáme z hlediska jeho významu, zejména zda neobsahuje objekt s nedostatečnou úrovní ACL, jehož změnou by mohl útočník program přesvědčit k provedení operace, kterou jeho vývojáři nezamýšleli. Také prozkoumáme síťovou komunikaci obou programů během jejich běhu. Ověříme jaké informace programy pomocí tohoto druhu komunikace získávají a jak obdržená data ověřují.

3.1 Kontrola souborů

V této sekci se zaměříme zejména na kontrolu PE souborů (soubory .EXE a .DLL). Prozkoumáme jejich hlavičky a ověříme, zda využívají mechanismy ASLR a DEP. K tomu využijeme nástroj CFF Explorer (1.3.2.4). V dalším

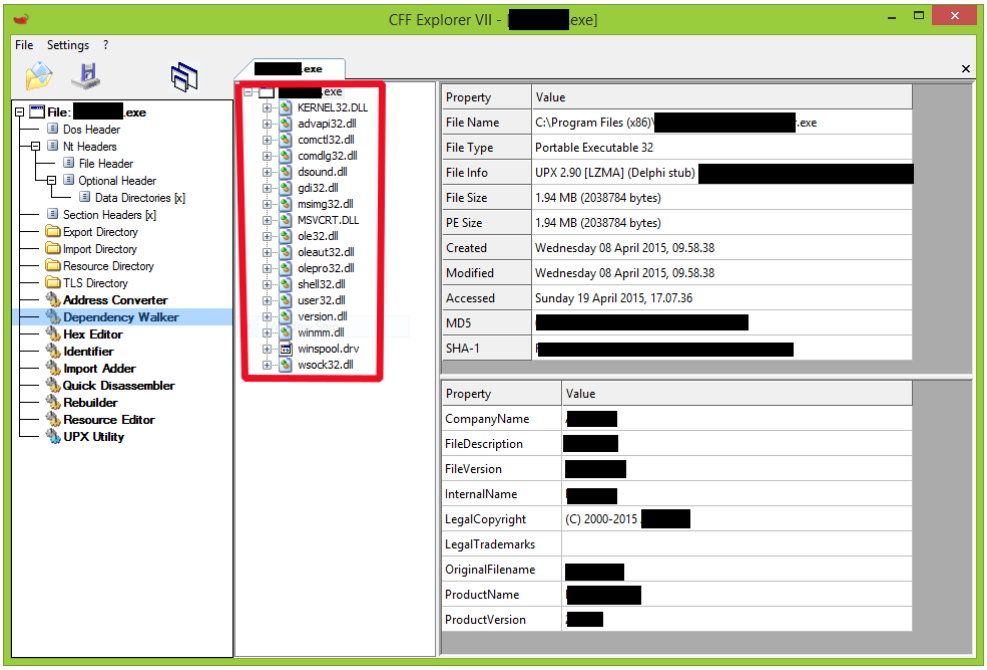
kroku je nutné získat seznam knihoven, které aplikace využívá (staticky i dynamicky linkovaných). Tento krok nám usnadní nástroje Strings 1.3.2.2 a opět CFF Explorer. Pomocí online databáze známých zranitelností [13] musíme ověřit, zda některá z knihoven v tomto seznamu neobsahuje již objevenou zranitelnost. Ta by mohla usnadnit případný útok na aplikaci. Protože s jednotlivými soubory úzce souvisí i jejich přístupová práva, definovaná pomocí ACL, měli bychom ověřit, zda jsou nastavena dostatečně s ohledem na význam a účel souboru.

3.1.1 Soubory aplikace Charlie

Aplikace Charlie má většinu souborů uloženou v adresáři *C:\Program Files (x86)*[redacted]*\Charlie*. Ke změně zde uložených dat je potřeba oprávnění systémového administrátora. To poskytuje vyšší bezpečnost, protože běžný uživatel nemá dostatečná oprávnění ke změně těchto dat. Zbytek souborů je umístěn v adresáři lokálního uživatele: *%USERPROFILE%\AppData... ...\Roaming*. Cílem tohoto adresáře je uložit data, která jsou specifická pro konkrétního uživatele a mohou se přesouvat spolu s uživatelským profilem [9]. Tato data jsou vlastněná uživatelem a ten má dostatečné oprávnění k jejich změně. I proto by zde uložená data neměla obsahovat volby, které mohou zásadním způsobem ovlivnit chod programu. Program Charlie do tohoto umístění ukládá informace o svém posledním běhu, uživatelské preference a další podobné informace. Proto můžeme konstatovat, že tento princip ctí.

Nejprve se zaměříme na přístupová práva dynamicky linkovaných knihoven. K tomu využijeme nástroj AccessEnum (1.3.2.1). Pokud by útočník mohl snadno takovou knihovnu zaměnit za jinou, která by kromě původních funkcí obsahovala i škodlivé funkce, jednalo by se o kritickou zranitelnost. Spustitelné soubory i dynamicky linkované knihovny programu Charlie jsou uloženy ve zmiňovaném adresáři *C:\Program Files (x86)*[redacted]*\Charlie*. Využívá se dostatečné úrovně zabezpečení pomocí ACL, protože útočník by ke změně zde uložených dat potřeboval oprávnění systémového administrátora.

V tomto adresáři je uloženo 6 spustitelných souborů a 4 dynamicky linkované knihovny. Pomocí nástroje CFF Explorer (1.3.2.4) jsme postupně prozkoumali jejich hlavičky. Pouze [redacted].exe využívá ASLR a DEP. Dále jsme postupně u každého PE souboru zjistili, které knihovny využívá. K získání informací o staticky linkovaných knihovnách jsme využili nástroje Binwalk (1.3.2.3) a Strings (1.3.2.2). K získání seznamu dynamicky linkovaných knihoven jsme použili program CFF Explorer. Ukázka použití tohoto nástroje je zobrazena na obrázku 3.1 — nalezené dynamicky linkované knihovny jsou v červeně zvýrazněné oblasti. V dalším kroku jsme se zjištěné knihovny prozkoumali v online databázi známých zranitelností [13]. Touto kontrolou nebyla objevena žádná známá zranitelnost.



Obrázek 3.1: Analýza dynamicky linkovaných knihoven

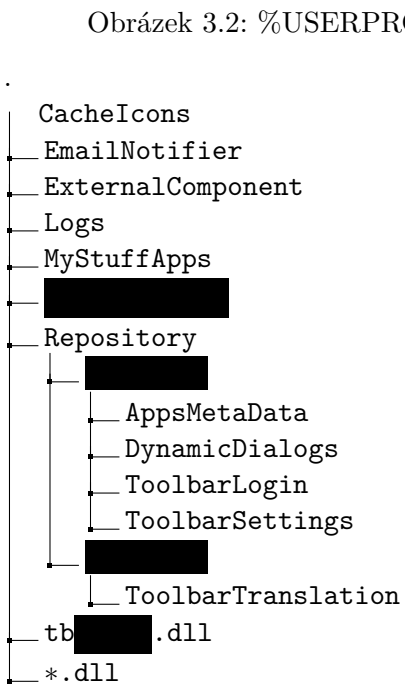
3.1.2 Soubory panelu

Panel do internetového prohlížeče má oproti samostatnému programu zejména tu nevýhodu, že počítá s úpravami vzhledu, které může provádět sám uživatel. Proto má většinu dat uloženou právě v jeho uživatelském prostoru. I právě proto by měl panel obsahovat mechanismus, který umožní kontrolu zpracovávaných informací.

Nejprve jsme pomocí nástroje AccessEnum (1.3.2.1) ověřili nastavení přístupových práv jednotlivých adresářů a souborů, které panel využívá. Adresář *C:\Program Files (x86)*[redacted]*\Panel* má přístupová práva nastavena, tak že ke změně jeho souborů je potřeba oprávnění systémového administrátora. V tomto adresáři je uložen pouze jeden spustitelný soubor. Zbylé soubory a adresáře panelu jsou uloženy v *%USERPROFILE%\AppData*. Ten budeme dále nazývat uživatelským prostorem. Upravovat zde uložená data může systémový administrátor či jejich vlastník. Tím je aktuální uživatel, kterému adresář *%USERPROFILE%* patří.

V adresáři *%USERPROFILE%\AppData\LocalLow*[redacted] je umístěna většina souborů panelu. Strukturu tohoto adresáře znázorňuje obrázek 3.2. Z něho je patrné, že v adresáři *...\LocalLow*[redacted] jsou uloženy i soubory obsahující nastavení panelu a několik DLL souborů. Protože ke změně těchto

Obrázek 3.2: %USERPROFILE%\AppData\LocalLow\[redacted]



dat není potřebné oprávnění systémového administrátora, zaměříme se v jedné z následujích sekcí i na kontrolu toho, jak panel zde uložená data ověřuje.

3.1.2.1 PE soubory

Nakonec jsme se opět zaměřili na kontrolu PE souborů. Pomocí nástroje CFF Explorer (1.3.2.4) jsme prozkoumali, jaké jsou využívány dynamicky linkované knihovny, a pomocí nástroje Binwalk (1.3.2.3) jsme získali seznam používaných staticky linkovaných knihoven. Následuje ukázka výpisu získaného použitím nástroje Binwalk na soubor [redacted].dll.

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------
0             0x0             Microsoft portable executable
317099        0x4D6AB         Copyright string: " 1995-1998
                              Mark Adler "
353456        0x564B0         Copyright string: " (c) 1992-2004
                              by P.J. Plauger, licensed
                              by Dinkumware, Ltd. AL licensed
                              by Dinkumware, Ltd.
                              ALL RIGHTS RESERVED."
363304        0x58B28             XML document, version: "1.0"
```

Binwalk ve svém výpisu zobrazuje pouze řetězce reprezentující jednotlivé copyrighty. Abychom získali informace o tom, k jakým knihovnám se vztahují, použili nástroj Strings (1.3.2.2). Ten ve svém výpise poskytuje informace o veškerých řetězcích z obyčejných znaků. Zaměřili jsme se na vyhledání copyrightů, které byly objeveny nástrojem Binwalk, a prozkoumání jejich okolí. V následující ukázce nezobrazíme výpis pro celý soubor [redacted].dll, ale jen okolí jednoho z copyrightů uvedených v předchozí ukázce.

```
+o*7
 inflate 1.1.3 Copyright 1995-1998 Mark Adler
HKEY
```

Takto získané názvy použitých knihoven jsme porovnali s online databází známých zranitelností [13]. Copyright uvedený v předchozí ukázce jsme objevili v několika souborech. Funkce *inflate* je součástí knihovny *zlib 1.1.3* a obsahuje známou zranitelnost s označením *CVE-2002-0059*. Jedná se o takzvanou *double free memory* zranitelnost, což znamená, že se knihovna jedno paměťové místo snaží uvolnit vícekrát bez předchozí opětovné alokace. Proti této zranitelnosti je známý útok, a pokud by potenciální útočník dokázal panel přinutit k provedení zranitelné funkce knihovny zlib, mohl by funkci inflate pomocí upravených dat přinutit k provedení jím poskytnutého kódu. Za zmínku stojí i fakt, že tato chyba byla objevena již v roce 2002, a přesto se takto zranitelná knihovna vyskytuje v několika DLL souborech včetně hlavního (ve verzi z roku 2014), který má na starosti ovládání panelu.

Pokud by některý z PE souborů obsahujících zranitelnou funkci *inflate* ASLR a DEP nevyužíval, mohl by potenciální útočník provést známý útok přímo. Používání ASLR a DEP v jednotlivých PE souborech jsme provedli pomocí nástroje CFF Explorer (1.3.2.4). V tabulce 3.1 jsou zobrazeny jednotlivé dynamické knihovny, ve kterých byl nalezen copyright zranitelné funkce *inflate*. Můžeme si povšimnout, že jedna z těchto funkcí nevyužívá ani DEP ani ASRL.

Tabulka 3.1: PE soubory obsahující zranitelnou funkci inflate

PE Soubor	**ASLR**	**DEP**
hk64[redacted].dll	Využívá	Využívá
hk[redacted].dll	Využívá	Využívá
ldr[redacted].dll	Nevyužívá	Využívá
prx[redacted].dll	**Nevyužívá**	**Nevyužívá**
[redacted].dll	Nevyužívá	Využívá
[redacted].dll	Nevyužívá	Využívá

3.1.3 Shrnutí sekce

U aplikace Charlie jsme neobjevili žádné nedostatky v logickém rozložení souborů ani v používaných knihovnách. Důležité PE soubory uloženy ve standardním adresáři *C:\Program Files* a jsou chráněny pomocí vyšší úrovně ACL — k jejich úpravě je nutné oprávnění systémového administrátora. Osobní nastavení uživatele je umístěno v adresáři, do kterého může zapisovat a data upravovat. Naopak nastavení, které ovlivňuje chování aplikace, je mimo dosah běžného uživatele. Spustitelné soubory a DLL soubory téměř výhradně nepoužívají ASLR a DEP (1.2.2.1), což by mohlo útočníkovi v případě další odhalené zranitelnosti usnadnit vývoj exploitu (zneužití této zranitelnosti). Protože jsme však zatím další zranitelnost neobjevili, můžeme tento nedostatek ohodnotit jako mírné riziko.

Panel naopak využívá pro uložení svých souborů téměř výhradně adresář *%USERPROFILE%\AppData\LocalLow*[redacted]. To znamená nedostatečné ACL k ochraně klíčových souborů, které jsou v něm s nejvyšší pravděpodobností umístěny. V jedné z dalších sekcí tento předpoklad ověříme. Panel ve svých DLL knihovnách opakovaně využívá knihovnu, ve které již byla nalezena zranitelnost. Jedna z DLL knihoven panelu, které mají tuto zranitelnou knihovnu staticky linkovanou, dokonce nevyužívá ani DEP ani ASLR. V případě, že by se podařilo panel přinutit provést činnost vedoucí k této chybě, mohl by potenciální útočník této známé zranitelnosti využít a spustit libovolný kód.

3.1.4 Síťová komunikace

Nyní prozkoumáme, zda a jak oba programy komunikují pomocí internetu. K této analýze jsme použili nástroj Wireshark (1.3.3.3). Pokusíme se odhalit zejména, zda programy přistupují k nějakým doménám automaticky, zda z nich stahují nějaké informace a zda k tomu využívají zabezpečené síťové protokoly. Automatické získávání dat pomocí síťové komunikace je nebezpečné zejména v tom, že nemusí být závislé na chování uživatele. Zranitelnost při tomto procesu by mohla způsobit, že se případnému útočníkovi podaří proniknout do počítače oběti, aniž by si toho oběť byla vědoma. Stejně jako v předchozí sekci rozdělíme poznatky podle toho, ke kterému ze zkoumaných programů patří.

3.1.5 Komunikace aplikace Charlie

Po spuštění samotné aplikace nedošlo k žádné síťové komunikaci. Totéž lze pozorovat pří průzkumu nabídky programu. Jedná se o poměrně očekávané chování, protože program ke svému běhu žádná další data nepotřebuje. Dále jsme se zaměřili na to, zda se program nepřipojuje k aktualizačnímu serveru, aby nám nabídl novou aktualizaci. Pokud by se jednalo o zabezpečené spojení s důkladným ověřováním dat, jednalo by se o velmi užitečnou funkci, zejména

protože by uživatele aplikace informovala o případné bezpečnostní aktualizaci. Tuto funkci aplikace Charlie bohužel ve své bezplatné verzi nepodporuje. Na druhou stranu se tím o něco snižuje prostor pro možný útok na aplikaci. Uživatel v případě zájmu o kontrolu dostupnosti aktualizace programu musí v nabídce kliknout na patřičnou položku, která ho zavede na webovou stránku. Ta pomocí parametrů o verzi používaného programu skrytých v URL zkontroluje, zda je dostupná nová verze aplikace Charlie. Kromě tohoto odkazu na aktualizace jsou v programu obsaženy i odkazy na webové stránky projektu, fórum a další webové servery. Všechny tyto odkazy používají nezabezpečený protokol HTTP.

Pokud programu poskytneme vstupní soubor, s kterým umí pracovat, tak se připojí k několika serverům a ověří, zda nejsou dostupné [redacted]. Ty nejsou pro běh programu potřebné, mohou však jeho používání velmi zpříjemnit. K ověření dostupnosti a případnému získávání [redacted] program využívá opět nezabezpečený protokol HTTP. Je proto nutné, aby byla takto stažená data důsledně kontrolována. K jejich získání aplikace Charlie využívá metodu POST protokolu HTTP a přistupuje na následující servery:

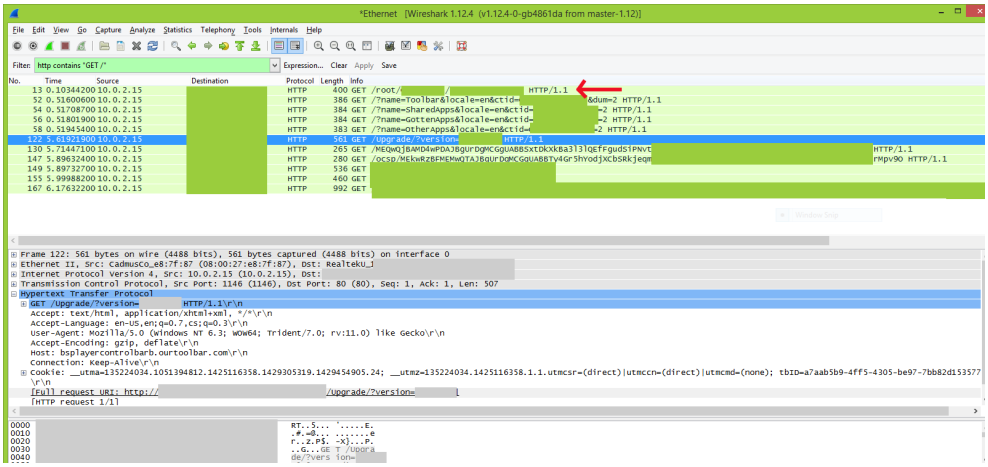
- s8.api.[redacted].com
- [redacted].api.[redacted].com
- api.get[redacted].com

Aplikaci Charlie můžeme také použít k [redacted]. V nabídce si vybereme položku, kterou chceme programem Charlie spustit. K dané URL program přistupuje pomocí protokolu HTTP.

3.1.6 Komunikace panelu

U panelu je situace odlišná. Ve svém výchozím nastavení totiž obsahuje i informace, které by se mohly zdát uživateli užitečné, ale s vlastním programem Charlie nijak nesouvisí. Jedná se například o informace o počasí či napojení na sociální síť Facebook. Bylo tedy dopředu jasné, že panel bude periodicky získávat alespoň některá svá data síťovou komunikací se vzdálenými servery. Dále jsme se opět zaměřili na způsob aktualizačního procesu panelu. Ten neprovádí kontrolu dostupných aktualizací automaticky, ale nabízí uživateli odkaz s kontrolou dostupnosti případné aktualizace. Odkaz využívá parametrů URL, podle kterých by měl ověřit aktuálnost používaného programu. Celou komunikaci jsme sledovali pomocí nástroje Wireshark (1.3.3.3). Na obrázku 3.3 je tento požadavek modře zvýrazněn. Tato funkce bohužel zcela nefunguje a uživatele informuje o dostupnosti nového programu i v případě, že skutečně využívá nejnovější verzi. Protože se nevyužívá šifrovaného spojení a panel nijak stažená data nekontroluje, jedná se o zranitelnost. Protože však stahování

těchto aktualizačních dat není prováděno automaticky, nejedná se o zranitelnost kritickou. Z dalšího pozorování vyplynulo, že panel periodicky, každé dvě hodiny, stahuje svůj konfigurační soubor. Tento soubor ovlivňuje vzhled i obsah panelu včetně odkazů, na které směřují jednotlivá tlačítka. Jedná se tedy o důležitý soubor, který by měl být panelem náležitě kontrolován. K jeho získání se využívá opět nezabezpečený protokol. Na obrázku 3.3 je tento požadavek označen červenou šipkou.



Obrázek 3.3: Obnovení panelu a ověření dostupnosti aktualizace

3.1.7 Shrnutí

Ověřili jsme, že aplikace Charlie po většinu času síťovou komunikaci nevyužívá. Jedinou výjimkou je po otevření vstupního souboru poskytnutého uživatelem. V tuto chvíli se aplikace připojí ke vzdáleným serverům a pomocí protokolu HTTP ověří dostupnost [redacted]. Pokud se podaří nějaký naleznout, tak se uživatele zeptá, zda ho má získat. Dále aplikace obsahuje několik odkazů, které využívají protokol HTTP. Program také umožňuje otevření vzdáleného souboru pomocí URL.

Panel komunikuje po síti mnohem více. To je poměrně očekávané chování, protože se jedná o součást internetového prohlížeče. Pravidelně získává různá data pro své součásti, jako jsou informace o aktuálním počasí. Panel opět obsahuje mnoho odkazů ve formě tlačítek. Všechny tyto odkazy využívají nešifrovaného protokolu HTTP. Pravidelně, vždy po uplynutí dvou hodin nebo na manuální žádost uživatele, je možné aktualizovat informace, které panel zobrazuje. Za tímto účelem panel získává vzdálený konfigurační soubor, který je umístěn na webu vydavatele panelu. K získání tohoto konfiguračního souboru je opět využíván protokol HTTP.

Lze tedy konstatovat, že oba programy – Charlie i panel – využívají síťovou komunikaci. V této je hojně využíván protokol HTTP, což usnadňuje případný útok. Mezi nejvíce kritickou komunikaci řadíme takovou, která probíhá bez vědomí uživatele. Protože takto komunikuje panel během získávání svého konfiguračního souboru, rozhodli jsme se tuto funkcionalitu panelu blíže prozkoumat.

3.2 Chování panelu

V předchozích částech této práce se nám podařilo odhalit, že panel k ukládání svých dat využívá adresář *%USERPROFILE%\AppData\LocalLow*[redacted]. Nastavení jeho přístupových práv umožňuje upravovat své soubory uživateli s oprávněním systémového administrátora a svému vlastníkovi. Zjistili jsme, že obsahuje několik DLL souborů a konfigurační soubory. Sledováním panelu pomocí nástrojů IDA Pro Free (1.3.2.7) a WinDbg (1.3.2.7) jsme pozorovali jeho chování. Ověřili jsme, že jsou tyto DLL soubory načteny do programu Internet Explorer a obsahují funkce panelu. Mezi ty patří získávání konfiguračního souboru, obrázků, načítání obrázku,.... Jedná se o zranitelnost, protože přístupová práva umožňují uživateli změnou souborů ovlivnit vzhled a chování panelu. Jak jsme již několikrát zmiňovali, každý program by měl ověřovat svá data. Protože však může být tento soubor snadno zaměněn, nic nebrání potenciálnímu útočníkovi soubor vyměnit za pozměněný. Stačila by mu k tomu znalost DLL souboru a nestřežená chvilka u počítače oběti. Ve stejném adresáři je umístěn i zbytek dat, které tato knihovna využívá. To znamená relativně slabé ACL u těchto souborů a o to důkladnější by měla být kontrola platnosti dat. Kontrolovat by se ideálně neměl jen formát a obsah dat ale i jejich původ.

Dále jsme zjistili, že panel periodicky získává svůj konfigurační soubor pomocí protokolu HTTP. Protože se jedná o soubor ovlivňující obsah tohoto panelu, zejména pak jeho tlačítka a odkazy, prozkoumáme blíže metodu získávání a ověřování tohoto souboru. Také se pokusíme prozkoumat strukturu tohoto souboru a zjistit, jakým stylem jsou upravovány jeho tlačítka. Pokud by mohl útočník upravit vzhled nějakého tlačítka a odkaz, na který toto tlačítko směřuje, pak by se jednalo o významnou bezpečnostní hrozbu. Z toho plyne, že se v této sekci postupně zaměříme na dvě témata — jakým způsobem s konfiguračním souborem panel zachází a ve druhé fázi na jeho strukturu.

3.2.1 Aktualizace konfiguračního souboru

Pomocí nástrojů WinDbg (1.3.2.7) a IDA Pro Free (1.3.2.7) jsme se zaměřili na to, jak panel aktualizuje svůj konfigurační soubor. Již dříve jsme zjistili, že tento soubor stahuje pomocí protokolu HTTP. Prozkoumáním registrových klíčů, které panel využívá, jsme zjistili, že URL k tomuto souboru je uložena

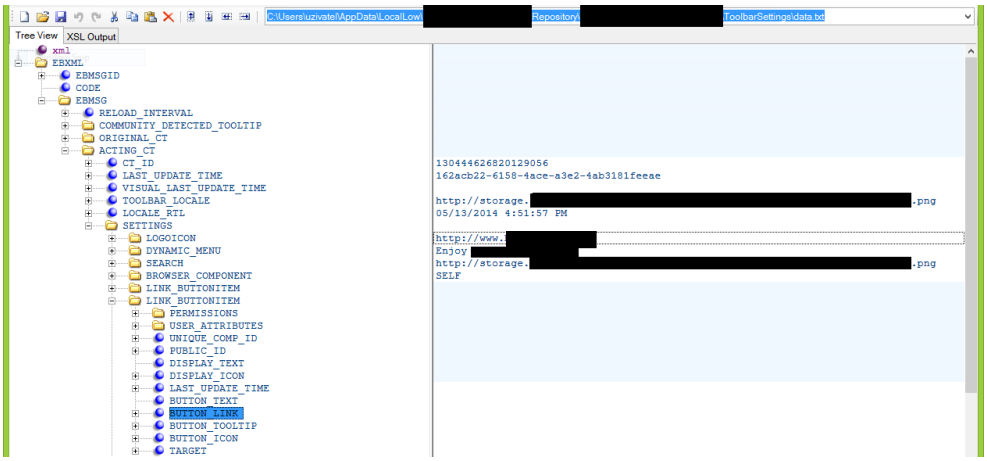
v následujícím registrovém klíči:
HKCU\Software\AppDataLow\Software[redacted]*\ToolbarSettings\ServiceUrl*
Slabinou tohoto registrového klíče je jeho ACL, protože k jeho úpravě není třeba oprávnění systémového administrátora. Stačí být přihlášen pod účtem uživatele, který bude panel využívat. Zároveň je v tomto klíči uložena URL, ze které je získáván konfigurační soubor. K jeho získání panel využívá protokol HTTP. Z těchto dvou faktů vyplývá, že útočník může panelu konfigurační soubor podvrhnout několika způsoby. Je proto nutné, aby panel obsahoval mechanismus kontrolující správnost zobrazovaných dat. Konfigurační soubor je potom uložen v uživatelském prostoru, konkrétně v adresáři: *C:\Users\uzivatel\AppData\LocalLow*[redacted]*\ToolbarSettings* s názvem „data.txt“. V tomto adresáři je uložen i soubor „data.bck.txt“, který podle svého názvu slouží jako záloha původního konfiguračního souboru.

Panel každé dvě hodiny nebo pomocí uživatelské volby aktualizuje zobrazované informace, například informace o počasí, obrázky tlačítek,. . . . Při každém svém obnovení se panel nejprve pokusí získat konfigurační soubor z URL umístěné v registrovém klíči. Tento soubor je stažen i v případě, že je shodný s aktuálně uloženým. Pouze v případě, že se toto stažení nepodaří, je původní konfigurační soubor ponechán. Abychom ověřili, zda panel ověřuje původ dat, zkopírovali jsme konfigurační soubor na testovací webový server a úpravou registrového klíče ho panelu podstrčili. Panel ho z testovacího serveru stáhnul a nahradil s ním původní verzi. Můžeme tedy předpokládat, že původ souboru není kontrolován. V dalším kroku jsme zkusili otestovat, co se stane, když soubor na testovacím webserveru přejmenujeme a adekvátně upravíme i URL v registrovém klíči. Panel se v takovém případě chová naprosto stejně, jako v prvním případě. V posledním kroku jsme zkusili v souboru naleznout známou URL a tu změnit na jinou. Takto upravený soubor jsme opět úspěšně načetli do panelu. To naznačuje chybějící mechanismy k ověření uložených dat. Tento předpoklad jsme potvrdili analýzou souboru [redacted].dll pomocí nástroje IDA Pro Free. V další podsekci tedy prozkoumáme strukturu tohoto souboru, abychom věděli, zda a jak ho může případný útočník pozměnit.

3.2.2 Struktura konfiguračního souboru

Nyní potřebujeme prozkoumat strukturu konfiguračního souboru, abychom mohli zjistit, které hodnoty by mohl útočník pozměnit, aby mohl panel využít ve svůj prospěch. Soubor má příponu TXT, a proto ho můžeme snadno prozkoumat v textovém editoru. Z prvního řádku souboru je vidět, že se jedná o XML soubor. Tvůrci souboru se nejspíše snažili ušetřit místo, a proto ze souboru odebrali veškeré formátování ve formě odsazení a odřádkování. K dalšímu zkoumání tohoto souboru jsme tedy použili nástroj XML Notepad 2007 [6]. Tento nástroj zobrazí XML souboru formou stromové struktury. Také umožňuje měnit obsah klíčů, hodnot i jejich parametrů a upravený soubor

uložit. Na obrázku 3.4 je zobrazena ukázka struktury konfiguračního souboru. Konkrétně jsme se zaměřili na nastavení jednoho tlačítka reprezentujícího odkaz.



Obrázek 3.4: Ukázka konfiguračního souboru panelu

Panel má tedy ve svém konfiguračním souboru nastaveny všechny prvky, které zobrazuje. Obsahuje adresu, na kterou tlačítko odkazuje a URL souboru, který reprezentuje obrázek zobrazovaný tlačítkem. Protože jsou tyto obrázky stahovány automaticky, rozhodli jsme se dále prozkoumat i mechanismus jejich ověřování. Panel používá k jejich vykreslování knihovnu GDI+, která je součástí operačního systému Windows. Nejprve obrázek stáhne a uloží do adresáře *%USERPROFILE%\AppData\LocalLow*[redacted]*\CacheIcons*, potom za příponu .png přidá ještě koncovku .tmp. V dalším kroku se pokusí obrázek otevřít pomocí funkce z knihovny GDI+. Pokud se otevření obrázku povede, pak koncovku .tmp odstraní a obrázek v adresáři ponechá, v opačném případě je soubor odstraněn. Při dalším obnovení panelu nejprve ověří, zda již adresář obrázek obsahuje. Pokud ano, pak ho zobrazí, jinak se ho opět pokusí stáhnout. Protože je k ověření obrázku využívána knihovní funkce operačního systému nedá se tento mechanismus jednoduše zneužít. Toto samozřejmě platí do doby, než by byla objevena zranitelnost této systémové knihovny. Ověřování toho zda je do panelu načítán obrázek, je tedy prováděno knihovní funkcí operačního systému. Panelu tedy nelze pomocí upraveného obrázku snadno podvrhnout závadná data, která by spustil.

3.2.3 Závěry z pozorování konfiguračního souboru

U konfiguračního souboru se výrazně projevuje špatný návrh adresářové struktury, která je panelem využívána. To samé platí i pro různé důležité registrové klíče. Protože jsou veškeré soubory i klíče uloženy v uživatelském

prostoru, mohou být upravovány i hodnoty důležité pro běh panelu. Toto se přímo dotýká konfiguračního souboru. Ten je totiž získáván pomocí URL uložené v registrovém klíči se slabým ACL (k jeho úpravě není potřeba administrátorského oprávnění) a poté uložen do adresáře, který má opět slabé ACL. Další slabinou je, že je tento soubor stahován pomocí nezabezpečeného HTTP spojení. Soubor tedy může být útočníkem snadno zaměněn i vzdáleně. Protože nedochází k naprosto žádné kontrole logického obsahu souboru, jedná se o poměrně významnou zranitelnost. Útočník sice nemůže aplikaci přimět k spuštění jiné aplikace či kódu, může však podvrhnout tlačítko, které panel zobrazí. Pokud bude toto tlačítko uživatele dostatečně dráždit, pravděpodobně na něj klikne. Poté bude přesměrován na útočníkův web, kde může očekávat skutečný útok, například formou stažení instalátoru škodlivého softwaru.

3.3 Shrnutí

V této kapitole jsme se zaměřili zejména na zkoumání běžného chodu aplikace Charlie i panelu, který je instalován do internetového prohlížeče. Nejprve si pro přehlednost shrneme zranitelnosti, které byly během této analýzy objeveny. Tabulka 3.2 obsahuje zranitelnosti objevené v samotném programu Charlie. Nejvýznamnější zranitelností objevenou přímo v této aplikaci, bylo stahování pomocných dat, které je realizováno pomocí nezabezpečeného HTTP protokolu. Absence automatických aktualizací výrazně snížila prostor pro případný útok. Přístupová práva souborů, které mohou ovlivnit chod aplikace Charlie, jsou nastavena tak, že je k úpravě těchto souborů nutné oprávnění systémového administrátora. Slabá ACL — tedy taková, že data může upravovat i obyčejný uživatel — jsou nastavena pouze u uživatelských [redacted] a nastavení vzhledu, což je očekávané chování. U žádné z knihoven (staticky i dynamicky linkovaných), které PE soubory aplikace Charlie používají, jsme v online databázi zranitelností [13] neobjevili žádnou zranitelnost.

Tabulka 3.2: Zranitelnosti aplikace Charlie

Zranitelnost	**Riziko**
Některé EXE a DLL soubory nepoužívají ASLR a DEP	Malé
V odkazech na weby využívá HTTP	Malé
K stahování pomocných dat používá HTTP	Střední

Tabulka 3.3 obsahuje zranitelnosti, které byly nalezeny v panelu internetového prohlížeče. Většina zde nalezených zranitelností vyplývá z nevhodného návrhu panelu – všechny soubory a registrové klíče jsou ukládány do uživatelského prostoru. Toto rozložení má za následek nedostatečné ACL u důležitých souborů a klíčů, které ovlivňují chování a vzhled panelu. Panel periodicky

získává svá data, přičemž však využívá protokol HTTP. Útočník může tedy posílaná data odchytit, pozměnit či zcela zaměnit za jiná. V panelu rovněž postrádáme i základní mechanismy kontroly obsahu získaného konfiguračního souboru.

Tabulka 3.3: Zranitelnosti panelu

Zranitelnost	**Riziko**
Slabé ACL u souborů a registrových klíčů panelu. Včetně podstatných souborů pro běh a funkce panelu.	Střední
Některé EXE a DLL soubory nepoužívají ASLR a DEP	Malé
Využití knihovny zlib 1.1.3 obsahující známou zranitelnost	Střední
Konfigurační soubor stahován pomocí HTTP	Střední
Odkazy poskytované panelem používají HTTP	Malé
Chybí ověřování konfiguračního souboru	Vysoké

Z těchto nalezených zranitelností považujeme za nejvýraznější hrozbu podvržení konfiguračního souboru panelu. Tento soubor je získáván zcela automaticky v periodických intervalech a umožňuje případnému útočníkovi upravit tlačítka s odkazy tak, jak si jen přeje. Tyto změny mohou předcházet další fázi útoku. Tou může být přesměrování na web útočníka, kde bude uživateli podvržen instalátor škodlivého softwaru nebo se tento web může tvářit jako přihlašovací stránka na některou ze známých služeb. Tento typ útoku se nazývá *phishing*. V další kapitole analyzujeme možnost, jak by mohl útočník konfigurační soubor panelu podvrhnout.

Kapitola 4

Návrh možných útoků a protiopatření

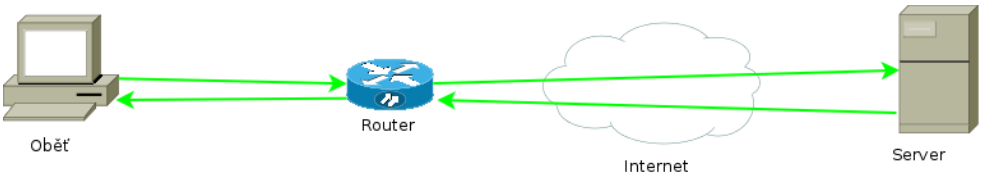
V předchozích kapitolách jsme identifikovali několik zranitelností, které s využíváním aplikace Charlie a jejího panelu souvisejí. Z nalezených zranitelností jsme vybrali dvě, které ohrožují uživatele nejvíce, a na ně bychom se měli v této kapitole zaměřit. Jedná se o zranitelnost instalačního procesu, kdy nejsou ověřována ani obdržená data ani identita serveru, ke kterému se aplikace připojuje. Druhou zranitelností, kterou se budeme v této kapitole zabývat, je zranitelnost panelu internetového prohlížeče. Rozebereme možnosti, jak by mohly být tyto zranitelnosti zneužity útočníkem a pokusíme se navrhnout protiopatření, která by tyto útoky co nejvíce omezila. V následující kapitole realizujeme ukázky těchto útoků v laboratorním prostředí, abychom demonstrovali, že se jedná o reálnou hrozbu. Tento přístup je nazýván spojením *Proof-of-concept*.

Mezi nejnebezpečnější typy útoků patří ty, které zneužívají komunikaci po síti. Útočník totiž během útoku nepotřebuje fyzický přístup k počítači oběti a ta si probíhajícího útoku nemusí být vědoma. Následkem útoku pak může být počítač oběti ovlivněn velmi dlouhou dobu, aniž by o tom oběť věděla. Protože obě vybrané zranitelnosti obsahují nešifrovanou komunikaci pomocí protokolu HTTP, pokusíme se v této kapitole navrhnout právě způsoby útoku na tuto komunikaci. Pro úplnost bychom měli zmínit, že počáteční fázi útoku, ve které se pokusíme nahradit část komunikace podvrženými daty, lze aplikovat na veškerou nezabezpečenou komunikaci. Útočník ani nutně nemusí znát obsah přenášených dat. Pokud je pouze v náhodných bitech pozmění, s nejvyšší pravděpodobností tím poškodí strukturu přenášených dat. Program se tato přenášená data v dalším kroku pokusí zpracovat, což se mu s vysokou pravděpodobností nepodaří. To může mít za následek nedostupnost některých služeb, které na této komunikaci závisí. V tomto případě si však oběť zpravidla ihned všimne, že něco není vpořádku. I proto se pokusíme útok provést tak,

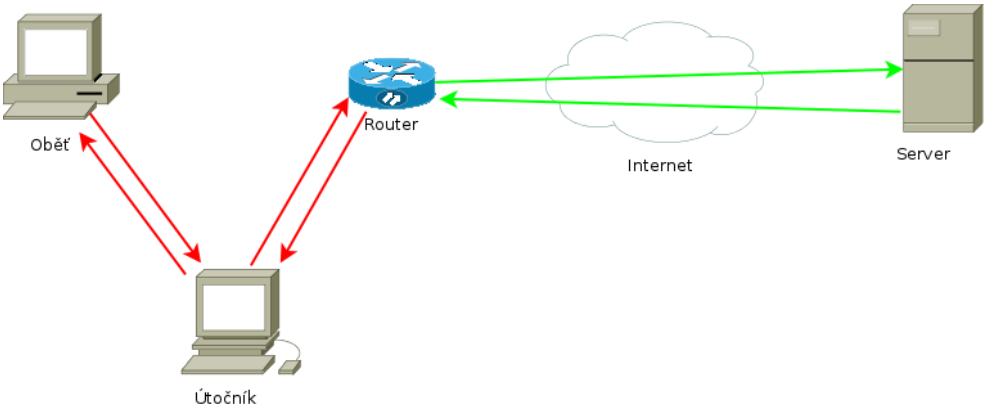
aby aplikace nedetekovala chybu a neposkytla tak oběti vodítko k odhalení probíhajícího útoku.

4.1 Možnosti útoku na nezabezpečenou komunikaci

Pokud chceme zaútočit na komunikaci, která probíhá mezi aplikací a jiným serverem, tak provádíme útok typu tzv. *Man-in-the-Middle* (MITM) [26]. Tento typ útoku znamená, že se útočníkovi podaří ovládnout část cesty, po které komunikace probíhá. Teoreticky takto může zneužít libovolnou část této cesty, ovšem v praxi je pro něho nejjednodušší ovlivnit komunikaci v lokální síti uživatele. Další body této trasy totiž spravují různí poskytovatelé internetového připojení a dá se očekávat, že mají vyšší zabezpečení. Schéma typického MITM útoku si můžeme prohlédnout na uvedených obrázcích. Obrázek 4.1 zobrazuje schéma běžné komunikace. Na obrázku 4.2 je zobrazena komunikace během útoku. MITM se projevuje tím, že útočník umístí své zařízení mezi počítač oběti a směrovač, který je s obětí ve stejné lokální síti. Veškerá komunikace mezi obětí a směrovačem tedy probíhá přes počítač útočníka. Obecně můžeme rozlišovat dva typy tohoto útoku.



Obrázek 4.1: Ukázka normální komunikace - server komunikuje s obětí přímo



Obrázek 4.2: Ukázka komunikace během MITM útoku - server s obětí komunikují přes útočníka

V prvním případě se jedná o útok, kdy je cílem útočníka nashromáždit co nejvíce informací o oběti. Pokud není cílem útočníka konkrétní osoba, pak je pro něho výhodné tento útok realizovat například ve veřejné Wi-Fi síti v obchodním centru, kde se připojuje velké množství uživatelů. Zde se anonymně připojí a pomocí tzv. *ARP Spoofing* útoku, který si blíže představíme později, všechna zařízení v této síti přesvědčí, aby komunikovala přes jeho zařízení. Poté jen přeposílá veškerou komunikaci a ukládá potenciálně zajímavé údaje. Mezi shromažďované data často patří přihlašovací údaje uživatelů.

V druhém případě se jedná o cílený útok, kdy útočník tuší, která data budou uživatelem ze vzdáleného serveru požadována. Protože uvažujeme situaci, kdy ovládl část trasy přenosu, a tato data jsou přenášena pomocí nešifrovaného HTTP připojení, může útočník tato přenášená data pozměnit tak, aby o tom uživatel nevěděl. Pro ilustraci uvedeme následující příklad. Oběť se rozhoduje, které oblečení má zvolit na cestu ven. Proto přistoupí na server s informacemi o počasí, kde ověří, jaká bude odpoledne teplota a zda bude pršet. Útočník dopředu zná strukturu odpovědi serveru s těmito informaci, a proto je snadno upravuje. Nic netušící oběť zobrazí takto získané informace a na jejich základě zvolí nevhodné oblečení. Uvedený příklad je relativně neškodný, nicméně útočník může tento princip využít k podvržení závažnějších dat.

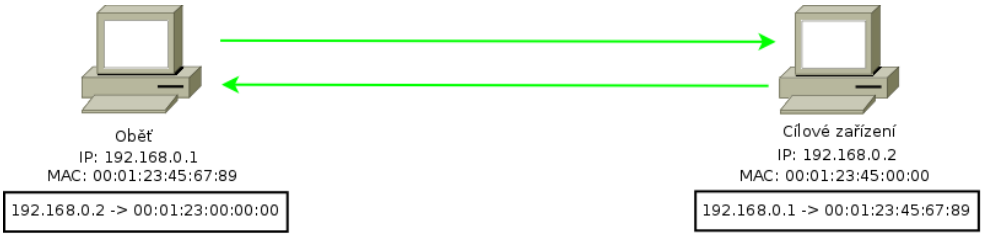
My se zaměříme na druhou zmiňovanou variantu, protože potřebujeme zkoumaným programům podvrhnout pozměněná data. V následujících podsekcích se seznámíme se známými metodami MITM útoku, které můžeme využít.

4.1.1 ARP Spoofing

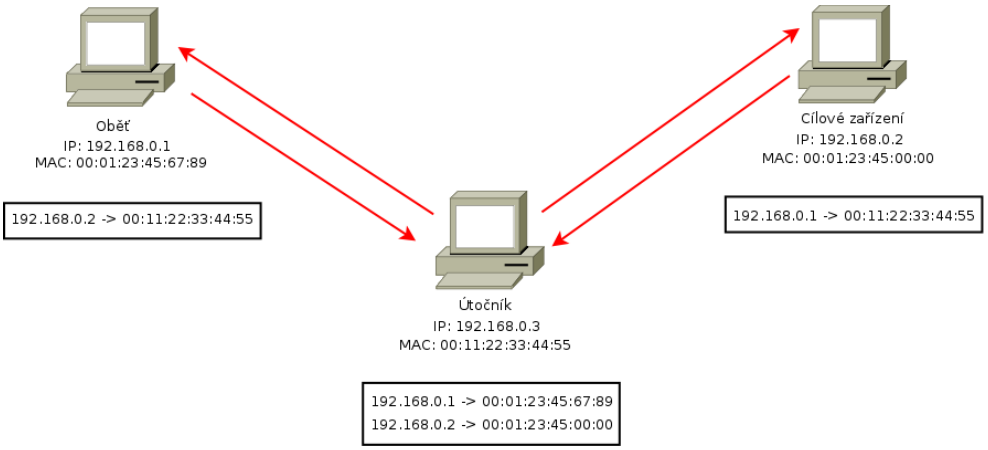
Abychom mohli vysvětlit princip tohoto útoku, musíme se nejdřív zaměřit na princip ARP protokolu. Nejprve stručně popíšeme princip komunikace, kterou aplikace Charlie a panel využívají. Aplikace jednotlivé servery, se kterými komunikuje, rozlišuje pomocí jejich doménových jmen. Doménová jména jsou překládána pomocí *Domain Name System* (DNS) protokolu na konkrétní IP adresy. Abychom mohli paket doručit na správné síťové rozhraní, je IP adresa přeložena na MAC adresu. K tomu se využívá protokol ARP (*Adress Resolution Protocol*) [25]. Aby se komunikace urychlila, tak každé zařízení obsahuje vlastní ARP tabulku. V té jsou uloženy známé překlady IP adres na MAC adresy. Pokud tedy chce zařízení komunikovat s konkrétní IP adresou, podívá se nejprve do své ARP tabulky. Pokud je v ní IP adresa obsažena, znamená to, že s ní v nedávné době komunikovalo, a proto je záznam aktuální. Na nalezenou MAC adresu poté odešle data. Pokud v tabulce není překlad k dispozici, tak nejprve provede broadcast ARP dotaz, která MAC adresa má danou IP adresu. Přijatou odpověď zařadí do své lokální ARP tabulky a data odešle na zjištěnou MAC adresu. Záznamy v lokální ARP tabulce

jsou aktualizovány i v případě, že zařízení data přijímá. V takovém případě informace získává z hlaviček přijatých dat.

Velkou výhodou tohoto principu je jeho jednoduchost a rychlost. Naopak nevýhodou je absence autentizace. Tohoto nedostatku může zneužít útočník. Pokud již zná IP adresu, se kterou chce oběť komunikovat, tak oběti zašle speciálně upravený paket. Ten v sobě obsahuje informaci o tom, že cílová IP adresa patří MAC adrese útočníka. Oběť tuto informaci zařadí do své lokální ARP tabulky a data odesílá útočníkovi. Útočník tedy „otrávil“ lokální ARP tabulku oběti — útok je proto často nazýván *ARP Poisoning* či *ARP Spoofing* [26]. Tímto způsobem může útočník přesměrovat komunikaci oběti na sebe. Pokud to samé provede i u cílového zařízení, stává se prostředníkem a může sledovat a přeposílat veškerou komunikaci mezi cílovým zařízením a obětí. Pro lepší ilustraci si můžeme situaci prohlédnout na přiložených obrázcích. Na obrázku 4.3 je zobrazena situace před provedením útoku. Na druhém obrázku 4.4 je znázorněna situace poté, co útočník úspěšně otrávil ARP tabulky obou zařízení. Pokud je cílovým zařízením výchozí brána lokální sítě, pak veškerá komunikace oběti s vnějším světem probíhá přes útočníka.



Obrázek 4.3: Ukázka normální komunikace včetně ARP tabulky



Obrázek 4.4: Ukázka komunikace během útoku včetně ARP tabulky

4.1.2 Útok na DNS

Podobně jako jsme v předchozí podsekci diskutovali útok na překlad IP adresy na MAC adresu, budeme se nyní zabývat možností útoku na překlad doménového jména na IP adresu [26]. DNS protokol je poměrně starý a bylo navrženo i realizováno mnoho útoků, které umožňují jeho zneužití. V posledních letech je snaha o rozšíření zabezpečené verze tohoto protokolu, která nese označení DNSSEC [2]. Ačkoliv tato verze eliminuje většinu útoků na překlad domén, není zatím v praxi velmi rozšířená. Dále se budeme zabývat klasickým DNS protokolem.

Pokud aplikace potřebuje získat IP adresu cílového serveru, tak odešle dotaz DNS serverům. V našem případě se jedná o dotaz typu A. Tento dotaz je většinou implementací realizován pomocí UDP rámce. DNS server hledanou adresu serveru zná, nebo se ptá nadřazeného serveru. U něho se situace opakuje. Dotaz se takto hierarchicky propracuje až k prvnímu serveru, který zná odpověď, a ta se vrací stejnou cestou. Aplikace, která se na počátku dotazovala, celou dobu čeká na odpověď. Poté co odpověď obdrží, použije adresu, kterou v této odpovědi nalezne. Pokud přijme odpověď od jiného serveru později, je ignorována.

Nyní budeme uvažovat situaci, kdy útočník úspěšně provedl „ARP Poisoning“ útok a komunikace oběti je tedy směrována přes jeho zařízení. Útočník pak jednoduše monitoruje probíhající komunikaci, a pokud zachytí DNS dotaz, zná veškeré jeho parametry. To mu umožňuje podvrhnout odpověď na konkrétní dotazy. Tato odpověď je obětí přijata a IP adresa dále použita. Jedinou možnou obranou před DNS útokem je prohlédnutí si této použité IP adresy. To ale téměř žádný běžný uživatel nedělá. Více detailů o DNS útocích si můžeme prohlédnout například v [23].

4.1.3 Shrnutí

V této sekci jsme se zabývali možnostmi útoku na síťovou komunikaci. Nejprve jsme se stručně seznámili s funkcí ARP protokolu, poté jsme diskutovali, jakým způsobem může být tento protokol zneužit útočníkem. Útočník může přesměrovat veškerou komunikaci oběti přes své zařízení, což mu umožňuje monitorování přes něho protékajícího síťového provozu, případně i provádění změn v přenášených datech. Tento útok může útočník realizovat například ve veřejné Wi-Fi síti. Pokud zvolí oblíbenou síť, například v obchodním centru, může nashromáždit komunikaci mnoha uživatelů. Obranou proti tomuto útoku může být například šifrování, protože útočník nemůže odchycená data snadno přečíst. Pro nás je tento útok zajímavý zejména v kombinaci s druhým útokem, který jsme si v této sekci představili. Jedná se o útok na protokol DNS, který je mimo jiné využíván k překladu doménových jmen na IP adresy.

Pomocí kombinace těchto útoků může útočník aplikaci Charlie či její panel přesvědčit, že požadovaná data se nacházejí na jiné (jím kontrolované) IP adrese. V dalších sekcích se budeme zabývat zejména tím, jaká data může útočník aplikaci Charlie či panelu podvrhnout, aby zneužil její zranitelnost.

4.2 Návrh útoku na instalační program

Z dříve učiněné analýzy vyplynulo, že instalační program aplikace Charlie během svého běhu stahuje dílčí instalační soubory pomocí nezabezpečené síťové komunikace. K získání IP adresy serveru s těmito instalacemi je využíván DNS protokol a data jsou dále přenášena pomocí nešifrovaného protokolu HTTP. Získaná data však nejsou žádným způsobem ověřována a jsou instalátorem rovnou spuštěna s právy systémového administrátora. Útočník by tedy mohl snadno nainstalovat do počítače oběti jinou aplikaci nebo dokonce škodlivý kód. Abychom demonstrovali nebezpečnost této zranitelnosti, pokusíme se instalátor přesvědčit, aby nainstaloval nejen požadovaný program, ale i další, o který oběť neměla zájem. Touto demonstrací ukážeme, že útočník získává možnost do počítače oběti nainstalovat libovolný program a zároveň svou činnost skrýt tím, že nainstaluje i požadovanou aplikaci Charlie.

Abychom instalačnímu programu dokázali vnutit námi požadovaná data, využijeme znalost URL, ze které jsou stahována. Další pokračování útoku bude uvažovat situaci, kdy je snadno realizovatelný „DNS Spoofing“. Například se jedná o situaci, kdy jsou útočník i oběť ve stejné lokální síti. Tato situace není nereálná, může se jednat například o společné využívání hotelové Wi-Fi sítě. Pomocí tohoto útoku se pokusíme instalátoru podvrhnout IP adresu našeho serveru, na kterém bude umístěn pozměněný instalační soubor. Cílem této úpravy bude vytvořit samorozbalovací balík SFX, ve kterém bude umístěn nejen instalační soubor programu Charlie, ale i další námi poskytnutý nástroj. Pokud se nám podaří takto připravený balík nainstalovat, aniž by si toho instalační program všiml, potvrdí se závažnost této zranitelnosti. Podobným způsobem by skutečný útočník mohl do počítače oběti nainstalovat zadní vrátka, která by umožňovala plné vzdálené ovládnutí tohoto počítače.

4.2.1 Závěr z návrhu útoku na instalační program

Protože instalační program zapisuje do chráněných adresářů, do nichž zápis vyžaduje oprávnění systémového administrátora, běží právě s těmito vysokými právy. Průběh instalace by tedy měl být vývojáři softwaru kontrolován o to důkladněji, protože diskutovaná zranitelnost znamená pro oběť ztrátu výhradního přístupu ke svým soukromým datům. V této sekci jsme se pokusili navrhnout modelový útok, jehož úprava by mohla mít právě takto rozsáhlé následky. Tento navrhovaný útok se skládá ze dvou částí. Nejprve připravíme

webový server, na který umístíme připraveny samorozbalovací balík typu SFX. Ten bude obsahovat samotnou aplikaci Charlie a námi poskytnutou aplikaci. V druhé fázi pomoci „DNS Spoofingu“ hlavnímu instalačnímu programu podsuneme takto připravená data a pokusíme se ho přesvědčit, aby je nainstaloval. Pokud se nám tento útok podaří realizovat, prokázali jsme závažnou zranitelnost instalačního programu.

4.3 Návrh útoku na panel internetového prohlížeče

Další závažné zranitelnosti jsme objevili u panelu internetového prohlížeče Internet Explorer. Odhalili jsme fakt, že tento panel obsahuje velké množství zranitelností pramenících z jeho špatného logického návrhu. Největším problémem jsou nedostatečná ACL u důležitých registrových klíčů a souborů. Všechny jsou umístěny v uživatelském prostoru daného uživatele, kde mohou být tímto uživatelem upravovány. Pokud tedy útočník získá přístup k počítači, do kterého je uživatel přihlášen, může tato data pozměnit. Tento panel však obsahuje i další zranitelnost, která znamená ohrožení uživatele. Jedná se o získávání konfiguračního souboru panelu z internetu pomocí nezabezpečeného spojení a bez následné kontroly původu a struktury dat. Tato zranitelnost je podstatná zejména z toho důvodu, že útočník nepotřebuje fyzický přístup k počítači oběti. Proto jsme ji identifikovali jako nejzávaznější z nalezených zranitenlostí. Abychom rizika s touto zranitelností spojené demonstrovali, navrhneme druhý modelový útok.

Druhý modelový útok bude směřovat právě na získávání konfiguračního souboru ovlivňujícího vzhled a obsah panelu. U útoku využijeme zejména fakt, že známe URL, ze které je získáván. Konfigurační soubor je z této URL stahován každé dvě hodiny a používá se k tomu nezabezpečený komunikační protokol HTTP. Získaný soubor není nijak kontrolován, a proto se pokusíme panel přesvědčit, aby použil námi upravenou verzi originálního souboru. K podvržení tohoto souboru provedeme „DNS Spoofing“ útok na známou URL. IP adresa, kterou takto panelu podsuneme, bude směrovat na webový server s tímto upraveným konfiguračním souborem. Pokud bude útok úspěšný, panel zobrazí upravené tlačítko. Za tímto tlačítkem se bude nacházet odkaz na námi zvolený web. Pokud by útočník dokázal oběť tlačítkem zaujmout, mohl by například tuto zranitelnost využít k phishingovému útoku. Pro naše testovací účely můžeme tento útok považovat za úspěšný ve chvíli, kdy v panelu upravíme tlačítko tak, aby směřovalo na námi zvolenou adresu.

4.3.1 Shrnutí

Nainstalovaný panel se stává součástí prohlížeče. Pokud ho uživatel určitou dobu používá, pak mu obvykle začíná důvěřovat. V případě, že bychom

úspěšně dokázali na panel zaútočit a upravit některé z jeho tlačítek, bylo by vysoce pravděpodobné, že by si oběť „nového“ tlačítka všimla. Toho by mohl chtít využít i potenciální útočník. V případě, že by si oběť tohoto tlačítka nejen povšimla, ale dokonce jí i zaujalo a rozhodla na něj kliknout, navštívila by odkazovaný web. Na tomto webu by mohly čekat další pasti, například phishingové stránky, které by útočníkovi umožnily získat citlivá data o uživateli. V následující kapitole se tedy pokusíme demonstrovat modelový příklad tohoto útoku. Pomocí „DNS Spoofing“ útoku zajistíme, že panel získá námi upravený konfigurační soubor. Útok budeme považovat za úspěšný, pokud bude na panelu zobrazeno upravené tlačítko, které bude směrovat na námi zvolenou URL.

4.4 Návrhy oprav nalezených zranitelností

Hlavním motivem pro tuto práci bylo zejména ověření bezpečnosti běžného uživatele během užívání aplikace Charlie a jejího panelu. V předchozích částech této práce se nám podařilo identifikovat několik různě závažných zranitelností. Na dvě z nich jsme dokonce navrhli možné útoky, které by vedly k vážnému bezpečnostnímu ohrožení uživatele. Následkem jednoho z těchto útoků by dokonce mohl útočník získat kontrolu nad počítačem oběti. Přitom by se mnohým z těchto zranitelností dalo relativně jednoduše předejít. V této sekci bychom chtěli provést návrhy, jak by se dalo těmto zranitelnostem vyvarovat, případně je opravit. Návrhy oprav provedeme postupně, jako jsme hledali zranitelnosti. Nejprve navrhneme opravy v instalačním procesu a poté se zaměříme na aplikaci Charlie a její panel.

4.4.1 Doporučení pro instalační proces

Výchozí konfigurace instalačního procesu by měla být co nejvíce omezená. To znamená, že při využití výchozího nastavení instalačního procesu, by měla aplikace nainstalovat pouze hlavní program bez volitelných částí. Tento přístup je někdy nazýván *secure by default* [5]. Pokud instalační proces ctí toto pravidlo, pak je ve výchozím nastavení velmi zúžen prostor, na který by mohlo být zaútočeno. U zkoumané verze instalačního programu toto pravidlo nebylo dodrženo a následkem bylo nainstalování panelu pro internetový prohlížeč, který obsahuje další zranitelnosti. Ukazuje se tedy, že vývojáři produktu upřednostnili množství poskytovaných funkcí na úkor bezpečnosti uživatele.

Nejčastěji opakovaným nedostatkem, který se opakuje ve všech zkoumaných částech, je využívání nezabezpečeného protokolu HTTP. Následkem toho není probíhající komunikace soukromá a obdržená data nelze považovat za důvěryhodná. Nasazení zabezpečeného protokolu HTTPS přitom není náročné a pro útočníka by bylo složitější přenášená data analyzovat či upravovat. Instalátor,

aplikace Charlie i panel by měli během komunikace ověřovat identitu serveru, s kterým komunikace probíhá. Tento krok lze realizovat například kontrolou certifikátů, které HTTPS využívá.

Poslední a nejpodstatnější doporučení se týká kontroly platnosti stažených dat. V případě instalačního programu jsou stahovány dílčí instalační programy, které jsou v dalších krocích nainstalovány hlavním instalátorem. Tyto dílčí instalátory jsou digitálně podepsány platnými certifikáty, které jsou podepsané důvěryhodnými certifikačními autoritami. Hlavní instalační program by tedy měl na základě certifikátu kontrolovat, zda do systému instaluje data podepsaná poskytovatelem aplikace Charlie.

4.4.2 Doporučení pro proces Charlie a panel

Nejvýznamnějším problémem, který jsme objevili přímo v aplikaci Charlie, bylo využívání nezabezpečeného protokolu HTTP. Stejný problém se nachází i v panelu prohlížeče, a proto můžeme následující doporučení směřovat na obě tyto aplikace. Použitím zabezpečené komunikace by případný útočník nemohl snadno analyzovat či podvrhnout přenášená data. Pomocí certifikátů, které protokol HTTPS využívá, by mohly aplikace ověřovat, zda komunikují s požadovanou protistranou.

Jak jsme již několikrát zmiňovali, mnoho zranitelností panelu pramení i z jeho špatného návrhu. Bylo by vhodné, aby panel lépe chránil své klíčové soubory a systémové registry, které se nacházejí v počítači uživatele. Mezi ně lze zahrnout například registrový klíč obsahující URL adresu umožňující získání nového konfiguračního souboru nebo pomocné DLL knihovny. Pokud by tyto soubory a registry byly chráněny pomocí přísnějšího ACL, nemohl by útočník v nestřeženém okamžiku u počítače oběti tyto hodnoty snadno ovlivnit. Zároveň však panel využívá soubory, které přímo bezpečnost panelu neovliňují, a ty by mohly mít stávající úroveň ACL. Bylo by vhodné, aby panel stažená data kontroloval. K tomu může využít například ověření identity serveru, z kterého jsou stažená, a nebo používat algoritmus RSA s veřejným klíčem serveru. Poslední doporučení se týká staticky linkovaných knihoven, které panel využívá v některých svých DLL souborech. Během analýzy jsme objevili využívání knihovny se známou zranitelností, která byla zveřejněná v registru zranitelností. Proto by bylo vhodné využívat aktuální verze knihoven třetích stran.

4.4.3 Shrnutí

V této sekci jsme se zaměřili zejména na možnosti oprav nalezených zranitelností. Obecně se dá shrnout, že nejvíce problémů bylo způsobeno využíváním nezabezpečené komunikace HTTP. Pokud by vývojáři nahradili tento protokol

jeho šifrovanou variantou HTTPS, mohli by pomocí ověření certifikátů snadno ověřovat identitu protistrany, se kterou komunikují. Tento protokol by také velmi znesnadnil MITM útoky, protože útočníkem odchycená data by byla v zašifrované podobě. Druhým častým problémem bylo chybějící ověřování obdržených dat. Zejména během instalace, kde jsou stažené instalátory podepsané platnými a důvěryhodnými certifikáty, by neměl být pro vývojáře problém jejich ověřování doplnit.

Zde bychom měli poznamenat, že v průběhu tvorby této práce byl zranitelný instalační soubor z webu projektu odstraněn. Je možné, že vývojáři aplikace Charlie některé ze zde uváděných zranitelností objevili. Instalátor byl na webu nahrazen souborem, který jsme v rámci této práce nazývali jako dílčí instalátor. Tento krok je významný pro bezpečnost uživatele, protože je eliminována nejvýraznější nalezená zranitelnost — chybějící ověřování získaných dat během instalace.

4.5 Závěr

Cílem této kapitoly bylo navrhnout teoretický postup, jak by mohl potenciální útočník nalezené zranitelnosti zneužít. Vytvořili jsme tedy základ, na kterém bude postavena následující, praktická kapitola. Nejprve jsme se zaměřili na možné útoky na síťovou komunikaci. Zde jsme se zaměřili především na situaci, kdy se útočník nachází ve stejné lokální síti jako oběť, která ke komunikaci s okolním světem využívá nezabezpečený protokol HTTP. Pro tuto situaci jsme si nejprve představili „ARP Spoofing“ útok, který útočníkovi umožňuje přesměrovat tok komunikace přes své zařízení. Dále jsme se zaměřili na „DNS Spoofing“, kdy útočník využívá předchozí metody k podvržení specifické DNS odpovědi.

Poté jsme navrhli útok na hlavní instalační proces aplikace Charlie, který stahuje dílčí instalace z internetu. Cílem takto navrženého útoku je demonstrovat nebezpečnost této instalace, kterou může útočník zneužít k instalaci libovolného nástroje do počítače oběti. Druhým útokem, který jsme v této kapitole navrhli, byl útok na získávání konfiguračního souboru ze vzdáleného serveru. U tohoto útoku nehrozí přímo získání kontroly nad zařízením oběti, může však být využit například k phishingu.

V poslední části této kapitoly jsme se zabývali možnostmi oprav jednotlivých zranitelností. Můžeme konstatovat, že využívání zabezpečené verze protokolu HTTPS ve spojení s ověřováním dat, by většinu z objevených zranitelností eliminovalo. U panelu jsme navrhli zejména logické rozdělení souborů na ty, které panel nezbytně potřebuje a na ty, které závisí na uživatelském nastavení.

První skupinu by bylo vhodné umístit do jiného adresáře s přísnějšími ACL, čímž by se předešlo jejich snadné úpravě běžným uživatelem.

V následující kapitole se pokusíme oba navržené útoky realizovat a tím názorně ukázat, že se jedná o reálnou hrozbu. Tuto realizaci provedeme v laboratorním prostředí — jejím prováděním neohrozíme běžné uživatele. Jedná se takzvaný *Proof-of-concept* útok.

Kapitola 5

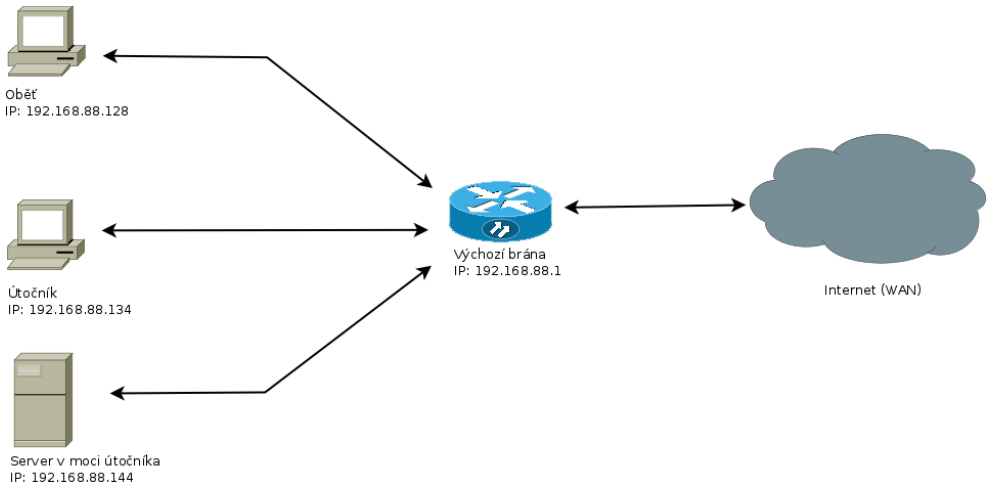
Realizace a testování

V následující části diplomové práce se zaměříme na realizaci dříve navržených modelových útoků a otestování jejich úspěšnosti v laboratorním prostředí. Zde se pokusíme simulaci situace, kdy se uživatel nachází ve stejné lokální síti jako útočník provádějící jednotlivé útoky. Hlavním cílem je prokázat nebezpečnost nalezených zranitelností. Pro lepší přehlednost tuto kapitolu rozdělíme na sekce, ve kterých se budeme věnovat dříve navrženým útokům. Nejprve se však seznámíme s testovacím prostředím, pak provedeme útok na instalátor aplikace Charlie a nakonec útok na panel.

5.1 Testovací prostředí

V této sekci si představíme testovací prostředí, ve kterém budou oba útoky realizovány. Hlavním motivem během vytváření tohoto prostředí bylo simulovat stav, kdy útočník sdílí s obětí stejnou lokální síť. To umožňuje sledovat komunikaci pomocí útoku typu MITM. Nejedná se o nereálnou situaci, protože mnoho uživatelů využívá frekventované Wi-Fi sítě, ve kterých je riziko odposlechnutí komunikace relativně vysoké. Pro názornost jsme do následující topologie zahrnuli i server, který útočník ovládá. V praxi tento server nemusí být součástí testovací sítě, ale může být umístěn za veřejnou IP adresou. Pro tuto práci byla zvolena varianta se serverem v lokální síti, abychom mohli veškeré prováděné útoky provádět v laboratorním prostředí. Tímto přístupem je zaručeno, že nebudeme nijak ohrožovat komunikaci běžných uživatelů. Použitou topologii si můžeme prohlédnout na obrázku 5.1.

Obětí budeme nazývat počítač, na který se pokusíme aplikovat navržené útoky. K tomuto účelu jsme připravili testovací počítač, na který jsme nainstalovali pouze operační systém Windows 8.1 s výchozí konfigurací. Simulujeme tak situaci, kdy má běžný uživatel zároveň oprávnění administrátora a může tak snadno nainstalovat libovolný software. Výchozí brána je hraniční směrovač, který je spojovacím bodem s vnějším světem. Útočníkem budeme



Obrázek 5.1: Testovací topologie

označovat počítač, který provádí jednotlivé útoky. Operační systém tohoto zařízení je GNU/Linux se zvolenou distribucí Fedora 21, což umožňuje využití linuxových nástrojů pro penetrační testy v počítačové síti. Posledním zařízením umístěným v lokální síti je server v moci útočníka. Na tomto zařízení je nainstalován webový server Nginx s daty, které se pokusí útočník podvrhnout v komunikaci oběti s protistranou ve vnějším světě. Operačním systémem tohoto zařízení je linuxová distribuce Arch Linux.

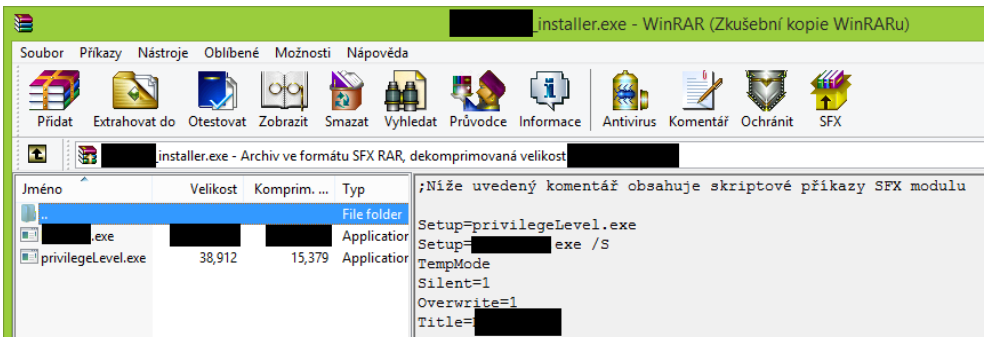
5.2 Realizace útoku na instalační proces

První navržený útok se pokouší zneužít síťové komunikace hlavního instalačního procesu. Realizovaným útokem se pokusíme prokázat zranitelnost tohoto procesu, která umožňuje útočníkovi upravovat instalovaná data. Následkem těchto úprav bude nainstalována nejen požadovaná aplikace Charlie se svým panelem, ale také bude spuštěn další software dle našeho výběru. Protože zároveň se spuštěním zvoleného nástroje nainstalujeme i obětí požadovanou aplikaci, je vysoká pravděpodobnost, že by si tohoto útoku oběť nevšimla. Skutečný útočník by navíc mohl místo běžné aplikace instalovat škodlivý software.

5.2.1 Příprava upravené dílčí instalace

Spolu s vlastní instalací programu Charlie, jsme se rozhodli spustit námi poskytnutý program. S tímto programem se blíže seznámíme v 5.2.1.1. Úspěšným spuštěním nástroje a ověřením toho, že byl spuštěn s právy systémového administrátora, demonstrujeme nebezpečnost instalačního procesu. Protože naším cílem bylo kromě samotné instalace aplikace Charlie spustit i námi

poskytnutný nástroj, rozhodli jsme se využít samorozbalovacího archivu. Ten se spuštěním automaticky rozbalí a pokud si to přejeme, provede předem definované operace. Tyto operace můžeme definovat pomocí speciálního skriptu. Samorozbalovací archiv i pomocný skript vytvoříme pomocí komerčního nástroje WinRAR, ve kterém všechny požadované volby provedeme pomocí grafického rozhraní. WinRAR [18] pak za nás vytvoří požadovaný skript. Námi provedenou konfiguraci si můžete prohlédnout na obrázku 5.2. K instalačnímu souboru aplikace Charlie jsme přiřadili přepínač „/S“, zajišťující jeho tichou instalaci. Dále jsme nastavili i tiché rozbalování archivu do dočasného adresáře. Výsledným chováním je, že se po spuštění archiv rozbalí a spustí (případně i nainstaluje) námi poskytnutý nástroj a nainstaluje aplikaci Charlie. Toto vše je provedeno bez toho aby se uživatele na něco zeptal.



Obrázek 5.2: Vytvořený SFX archív

5.2.1.1 Spouštěný program

Nyní se seznámíme s programem, který bude spuštěn spolu s instalací vlastní aplikace Charlie. Tento program zaznamená do pomocného souboru informace o tom, s jakou úrovní oprávnění byl spuštěn [29]. Tento pomocný soubor je uložen na plochu uživatele. Následuje ukázka zdrojového kódu, který je využit k získání informací o úrovni oprávnění, se kterými byl proces spuštěn:

```
success = OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY,
                                                    &hToken);

if(success)
        success = GetTokenInformation(hToken,
                        TokenElevationType,
                        out_elevation_type,
                        sizeof(*out_elevation_type),
                        &cbSize);
```

Detailní zdrojový kód můžeme prohlédnout na přiloženém CD — v předchozí ukázce jsme vynechali příkazy, které souvisejí se zpracováním případných chyb. Po provedení operací z předchozí ukázky je uložena v proměnné *out_elevation_type* informace o tom, s jakými právy byl proces spuštěn. Touto hodnotou může být:

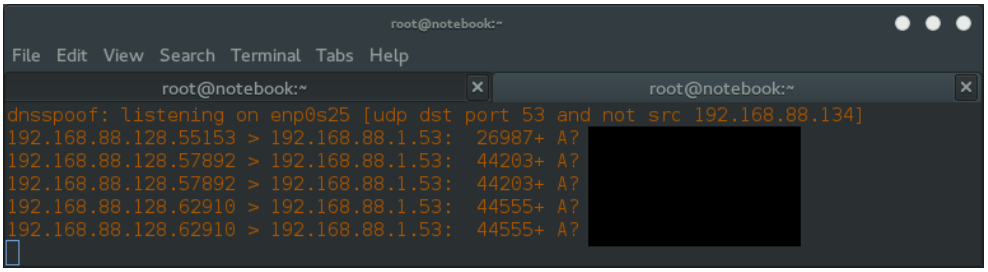
- **TokenElevationTypeDefault**
  Práva běžného uživatele, nemůže získat administrátorská práva
- **TokenElevationTypeFull**
  Administrátorské oprávnění
- **TokenElevationTypeLimited**
  Práva běžného uživatele, může získat administrátorská práva

Dále program získá a podrobně vypíše jednotlivá privilegia, se kterými je spuštěn. Program nejprve načte informace o Tokenu procesu, ze kterého získá jeho privilegia a nakonec i jejich názvy. Zjednodušenou verzi využitého kódu si můžeme prohlédnout v následující ukázce:

```
OpenProcessToken(GetCurrentProcess(), TOKEN_QUERY, &token)

DWORD length = 0;
GetTokenInformation(token, TokenPrivileges, NULL, 0, &length

TokenInfo = new char[length];
GetTokenInformation(token, TokenPrivileges, TokenInfo,
               length, &length)

TOKEN_PRIVILEGES* privs = (TOKEN_PRIVILEGES*)TokenInfo;


for (DWORD index = 0; index < privs->PrivilegeCount; index++)
{
        DWORD dwSize = 0;
        LookupPrivilegeName(NULL, &privs->Privileges[index].Luid,
                        NULL, &dwSize);
        LPTSTR szName = new TCHAR[dwSize + 1];
        LookupPrivilegeName(NULL, &privs->Privileges[index].Luid,
                        szName, &dwSize);
        file << szName << std::endl;
        delete[] szName;
}
```

5.2.2 Síťová komunikace

Druhým krokem v tomto útoku bylo podvržení dat oběti. Protože známe URL (http://www.[redacted]_installer.exe), ze které je získávána dílčí instalace, rozhodli jsme se využít útok „DNS Spoofing“ v kombinaci s „ARP Poisoning“ útokem. Nejprve jsme pomocí nástroje Ettercap [16] realizovali MITM útok pomocí metody „ARP Poisoning“. Protože nyní probíhala veškerá komunikace oběti přes útočníka, mohli jsme realizovat „DNS Spoofing“ útok na požadovanou URL. K tomuto útoku jsme využili nástroj dnsspoof z balíku programů dsniff [24]. Obrázek 5.3 ukazuje záznam tohoto útoku, který byl generován nástrojem dnsspoof. Z obrázku je patrné, že nástroj odchytil několik DNS dotazů typu A, jejichž odpověď pozměnil. Vstupem do nástroje dnsspoof je námi vytvořený konfigurační soubor *hosts.txt*, který obsahoval jediný řádek: 192.168.88.144 www.[redacted].com. Tímto jsme zajistili, že pokud se oběť zeptá DNS serveru na IP adresu dané URL, přesměrujeme ho na server, který je pod naší kontrolou.



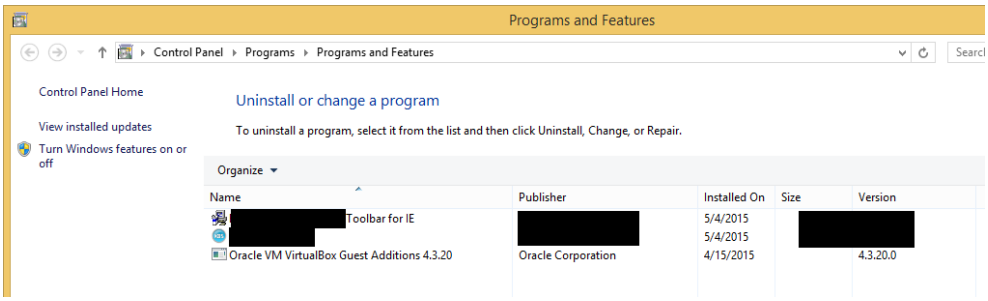
Obrázek 5.3: Útok pomocí nástroje Dnsspoof

5.2.3 Útok z pohledu oběti

V předchozích částech této sekce jsme se seznámili s úhlem pohledu útočníka, který celý proces podvržení dat oběti připravil. Z bezpečnostního hlediska je však pro nás mnohem více zajímavější pohled oběti. Ta je v situaci, kdy se například na dovolené rozhodla [redacted], a proto si pomocí hotelové Wi-Fi chce nainstalovat aplikaci Charlie. Aby byl útok úspěšný, neměla by oběť pojmout podezření, že byl do jejího počítače nainstalován i jiný nástroj.

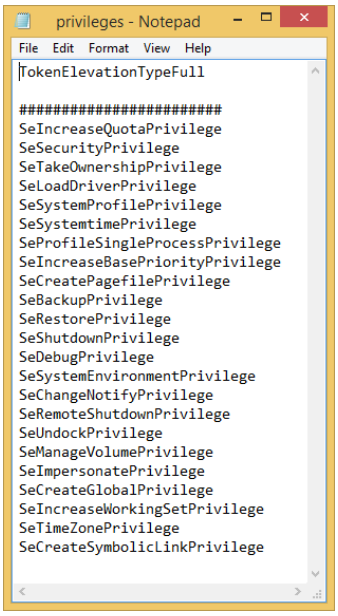
Nic netušící oběť spustila síťovou instalaci aplikace Charlie, která je podepsána platným certifikátem, a dialogové okno „Řízení uživatelských účtů“ vydavatele softwaru označilo za důvěryhodného. Dále oběť potvrdila přednastavené volby a spustila instalaci. Ta úspěšně proběhla a uživatel byl informován o jejím úspěchu. Nebyl tedy důvod pojmout jakéhokoliv podezření, že byl v rámci instalace spuštěn i jiný nástroj. Pro kontrolu, že byly nástroje doopravdy nainstalovány, se zaměřme na obrázek 5.4, reprezentující stav in-

stalovaných programů v zařízení oběti. Jak vidíme, instalací byla úspěšně nainstalována aplikace Charlie i panel.



Obrázek 5.4: Instalované programy - po útoku

Zároveň však byl úspěšně spuštěn i nástroj, který byl součástí samorozbalovacího archivu. Ten zaznamenal informace o úrovni oprávnění, se kterými byl spuštěn. Tyto informace byly zapsány do pomocného souboru spolu s privilegii, které proces obdržel. Výsledek si můžeme prohlédnout na obrázku 5.5. Z toho je patrné, že proces byl spuštěn s oprávněním systémového administrátora a zároveň s mnoho privilegii. To znamená, že byl útok úspěšně realizován.



Obrázek 5.5: Privilegia spuštěného procesu

5.2.4 Shrnutí

Pomocí nástroje WinRAR jsme připravili samorozbalovací archiv SFX, který instaloval nejen aplikaci Charlie, ale spustil námi poskytnutý nástroj. Dále jsme pomocí specializovaných nástrojů úspěšně realizovali MITM útok a oběti tento SFX archiv podvrhli místo originálních dat. Instalační proces tuto záměnu nijak nedetekoval a archiv spustil. Na závěr oběť informoval o úspěšné instalaci aplikace Charlie. Z výstupu námi poskytnutého programu jsme ověřili, že byl spuštěn s oprávněním systémového administrátora. Podařilo se nám tedy potvrdit významnou zranitelnost instalačního procesu, která může mít za následek ovládnutí počítače oběti.

V průběhu tvorby této diplomové práce tuto zranitelnost nejspíše objevili i autoři aplikace Charlie, a proto tento závadný instalační program ze svého webu stáhli a nahradili ho „klasickým“ instalačním programem, který ke svému běhu nepotřebuje internetové připojení. Do budoucna tedy bude zajímavé sledovat, zda další verze této aplikace budou síťovou instalaci využívat nebo se vývojáři rozhodli jít cestou lokální instalace. Pokud by tomu tak bylo, tak by tato největší zranitelnost byla eliminována.

5.3 Realizace útoku na panel

Druhým útokem, který jsme se rozhodli v této práci vyzkoušet, byl útok na doplňkovou komponentu, která byla doplněna instalačním programem v průběhu instalace. Jedná se o panel pro internetový prohlížeč Internet Explorer. U tohoto panelu jsme objevili vícero zranitelností. Rozhodli jsme se demonstrovat útok na získávání konfiguračního souboru, protože se jedná o síťovou komunikaci a útočník tedy nepotřebuje fyzický přístup k zařízení oběti. Cílem navrženého útoku je podvrhnout panelu upravený konfigurační soubor, který způsobí zobrazení pozměněného tlačítka, za kterým se bude skrývat námi zvolený odkaz. Útočník by mohl takovou zranitelnost využít k provedení další fáze komplexního útoku. Například by mohl oběť přesvědčit, že se za tlačítkem skrývá odkaz na internetové bankovnictví. V případě, že by dokázal tyto stránky věrohodně napodobit a poté, co by oběť zadala přihlašovací údaje, by byla přesměrována na pravé internetové bankovnictví, získal by útočník tyto přihlašovací údaje oběti. Tomuto napodobení existujícího webu se říká *phishing*.

5.3.1 Příprava konfiguračního souboru

Prvním krokem tohoto útoku je vytvoření upraveného konfiguračního souboru pro panel internetového prohlížeče. Se strukturou tohoto panelu jsme se blíže seznámili dříve a této znalosti nyní využijeme. Jedná se o XML soubor, který obsahuje URL adresy pro získání obrázku tlačítka a URL pro ukrytý

odkaz. Protože známe URL originálního souboru, můžeme ho snadno získat a pozměnit. V originálním konfiguračním souboru jsme nalezli část reprezentující tlačítko s odkazem. Tuto část jsme upravili, abychom dosáhli požadovaných vlastností. Nyní si můžeme prohlédnout provedené změny:

```
<LINK_BUTTONITEM>
        <PERMISSIONS>
        <EDIT>True</EDIT>
        <MOVE>True</MOVE>
        <DELETE>True</DELETE>
    </PERMISSIONS>
    <USER_ATTRIBUTES>
        <PERMISSION>FULL</PERMISSION>
        <SHOW_IN_CHEVRON>True</SHOW_IN_CHEVRON>
    </USER_ATTRIBUTES>
    <UNIQUE_COMP_ID>130444626819505051</UNIQUE_COMP_ID>
    <PUBLIC_ID>328676a1-3f69-4042-aeff-5b6fe85a67b6</PUBLIC_ID>
    <DISPLAY_TEXT />
    <DISPLAY_ICON>http://192.168.88.144/button.png</DISPLAY_ICON>
    <LAST_UPDATE_TIME>05/13/2014 4:51:57 PM</LAST_UPDATE_TIME>
    <BUTTON_TEXT />
    <BUTTON_LINK>http://192.168.88.144/hacked.html</BUTTON_LINK>
    <BUTTON_TOOLTIP>Hacknuto</BUTTON_TOOLTIP>
    <BUTTON_ICON>http://192.168.88.144/button.png</BUTTON_ICON>
    <TARGET>SELF</TARGET>
</LINK_BUTTONITEM>
```

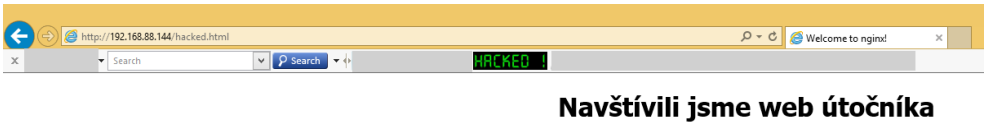
Jak si můžeme v uvedené ukázce povšimnout, panel použije námi zvolený obrázek typu PNG na zobrazení tlačítka. Tento obrázek jsme upravili tak, aby rozměrově odpovídal původnímu. Dále jsme k tlačítku doplnili popis „Hacknuto“ a odkaz přesměrovali server, který je v naší moci.

5.3.2 Síťová komunikace

Abychom tento konfigurační soubor mohli podsunout oběti, opět jsme využili předpokladu, že se oběť i útočník nacházejí ve společné lokální síti. Provedli jsme MITM útok pomocí „ARP Spoofing“ metody, na který navazoval DNS Spoofing URL (http://settings.toolbar.[redacted].com), kterou panel využívá k získání konfiguračního souboru. Druhá část, tedy „DNS Spoofing“, byla realizována pomocí nástroje dnsspoof. Tomuto nástroji jsme opět poskytli jednořádkový vstupní soubor „hosts.txt“, tentokrát s následujícím obsahem: 192.168.88.144 settings.toolbar.[redacted].com. Tímto jsme zajistili podvržení konfiguračních dat.

5.3.3 Útok z pohledu oběti

Pokud oběť již má internetový prohlížeč spuštěný, načte se konfigurace až po uplynutí dvouhodinového časového úseku. Při novém spuštění prohlížeče či jeho záložky je však do tohoto aktuálního okna načten panel znova. To má za následek stažení konfiguračního souboru a zobrazení požadovaných dat. V průběhu útoku panel z počátku nedokázal z nám neznámého důvodu správně zobrazit obrázek tlačítka (odkaz za ním byl nastaven správně). Po otevření a opětovném zavření několika záložek bylo tlačítko načteno a od té doby bylo zobrazováno správně. Výsledek útoku si můžeme prohlédnout na přiloženém obrázku 5.6. V panelu oběti bylo úspěšně pozměněno tlačítko, které směřovalo na web útočníka.



Obrázek 5.6: Ukázka panelu po druhém útoku

5.3.4 Shrnutí

I ve druhém útoku se nám podařilo úspěšně demonstrovat nebezpečí plynoucí z nalezené zranitelnosti. Celý útok proběhl podvržením síťové komunikace a využíval naprosto chybějícího mechanismu ověřování obdržených dat. Tentokrát se nejedná o zranitelnost, která vede k přímému ovládnutí počítače oběti útočníkem, ovšem i kvůli možnosti dalšího útoku, který může být skryt za odkazem v upraveném tlačítku, ji považujeme za významnou.

5.4 Závěr

V testovacím prostředí jsme vyzkoušeli oba dříve navržené útoky. Testovací prostředí napodobovalo využití veřejné Wi-Fi sítě, kdy se oběť i útočník nachází ve stejné lokální sítí. Názorně jsme si ukázali, jak je v takovém případě nebezpečné použití nezabezpečené síťové komunikace, kdy můžeme pozměňovat přenášená data. U obou útoků jsme úspěšně podvrhli námi upravená data, aniž by to zkoumaný proces detekoval. Následkem bylo výrazné ohrožení bezpečnosti uživatele. To v prvním případě může vést až k získání kontroly nad počítačem oběti. Ve druhém demonstrovaném útoku jsme kontrolu nad uživatelovým počítačem nezískali. Ukázali jsme však riziko, které z používání tohoto panelu vyplývá.

Obecně můžeme konstatovat, že obě prokázané zranitelnosti znamenají nezanedbatelné ohrožení uživatele. Ten by tedy panel pro internetový prohlížeč ani testovanou síťovou instalaci neměl využívat. U samotné aplikace Charlie jsme takto výrazné zranitelnosti neobjevili, nicméně program musíme hodnotit jako celek, a to včetně jeho instalačního procesu a nainstalovaných doplňků. Proto můžeme konstatovat, že jsme objevili významné zranitelnosti s aplikací Charlie úzce spojené. Pro úplnost musíme dodat, že i ona využívá ke svému běhu nezabezpečenou komunikaci, což jak jsme si ukázali, přináší další rizika. Obsahuje například odkaz na web s případnou aktualizací. Tento odkaz nevyužívá HTTPS a uživatel tak nemá jistotu, zda jsou získaná data platná.

Závěr

V rámci této práce jsme se měli seznámit s metodami a programy, které jsou využívány v rámci zpětného inženýrství. Takto nabyté znalosti jsme měli použít k analýze aplikace, kterou jsme společně s vedoucím práce vybrali. U této analýzy jsme se zaměřili zejména na prozkoumání zranitelností vyplývajících z komunikace aplikace s okolním prostředím. Nalezené zranitelnosti jsme měli zdokumentovat a provést návrh modelového útoku a možných protiopatření, která by tyto zranitelnosti eliminovaly. V poslední části této práce jsme měli dokázat reálnou hrozbu útoku pomocí názorné ukázky v testovacím prostředí. Protože je tato práce veřejně dostupná a zároveň obsahuje popis a názornou demonstraci útoku na testovanou aplikaci, bylo nutné zakrýt veškerá vodítka, která by mohla znamenat identifikaci zkoumané aplikace a následné zneužití práce k provedení útoku.

V úvodu této práce jsme se krátce vrátili k volbě testované aplikace a zdůraznili jsme, že vzhledem k její rozšířenosti by mohly její zranitelnosti znamenat výrazné bezpečnostní riziko pro širokou masu uživatelů. Abychom předešli zneužití této práce, tak jsme pro aplikaci zvolili pseudonym Charlie. Také jsme se zabývali celým procesem bezpečnostní analýzy libovolné aplikace. Rozebrali jsme důvody, proč musí libovolný program důkladně ověřovat veškerou komunikaci se svým okolím, a uvedli nejčastější možnosti, jak tato komunikace probíhá. V závěru této úvodní části jsme diskutovali základní metody zpětného inženýrství, které lze k bezpečnostní analýze využít, a doplnili jsme i stručné seznámení s některými možnými nástroji.

Po tomto teoretickém úvodu jsme se zabývali instalací aplikace Charlie. Zjistili jsme, že se jedná o síťovou instalaci programu, která umožňuje zvolit, zda chceme nainstalovat i panel pro internetový prohlížeč. Protože byla tato možnost nastavena jako výchozí, pokračovali jsme v instalaci s tímto nastavením. Instalační proces dále stahuje dílčí instalátory pomocí nezabezpečeného protokolu HTTP. Protože tento protokol umožňuje snadné podvržení

dat, rozhodli jsme se na získávání instalací zaměřit detailněji. Podařilo se nám zjistit, že ačkoliv jsou oba dílčí instalační programy podepsány platným certifikátem, hlavní instalační program neprovádí jejich ověření.

V dalším kroku jsme se zaměřili na bližší analýzu chování vlastní aplikace Charlie. Největším problémem, který jsme u této aplikace objevili, bylo opět využívání nezabezpečeného protokolu HTTP k získávání pomocných dat. Používání tohoto protokolu se ukázalo být nešvarem, který se opakoval u všech zkoumaných částí. Panel pro internetový prohlížeč, který byl spolu s aplikací nainstalován, tento protokol využívá i k získávání konfiguračních souborů. Po bližším prozkoumání tohoto mechanismu jsme opět objevili chybějící ověření obdržených dat. U panelu se projevil i problém s logickým návrhem tohoto panelu, který se projevil v nedostatečné úrovni ACL.

Abychom mohli navrhnout modelové útoky na nalezené zranitelnosti, provedli jsme v této části diplomové práce diskuzi možných útoků na síťovou komunikaci. Zabývali jsme se především útokem typu „ARP Poisoning“ a na něho navazujícím „DNS Spoofing“ útokem. Tím jsme si ukázali, jak může útočník oběti poskytnout pozměněná data. Díky tomuto základu jsme mohli navrhnout útok na instalační proces. Jeho cílem bylo nainstalovat do počítače oběti mimo požadovanou aplikaci i software, který si zvolí útočník. Takto by mohl získat kontrolu nad počítačem oběti. Druhý navržený útok spočíval v záměně konfiguračního panelu internetového prohlížeče. Následkem toho uživatel spatří ve svém prohlížeči nezvyklé tlačítko, které ho přesměruje na web útočníka.

Po návrhu útoků jsme provedli návrh úprav aplikace a jejích komponent, které by omezily či úplně odstranily nalezené zranitelnosti. Společným prvkem pro všechny zkoumané části bylo doporučení využívání zabezpečeného protokolu HTTPS, který by utajil přenášená data a umožnil kontrolovat identitu protistrany. Dále jsme doporučili ověřování přijatých souborů a v případě panelu umístění klíčových souborů do adresáře s přísněji nastavenými ACL.

V závěru práce jsme v testovacím prostředí vyzkoušeli provedení vytvořených návrhů útoků. Nejprve jsme popsali využitou testovací topologii. Ta simulovala situaci, kdy oběť využívá stejnou lokální síť jako útočník. V této síti jsme využili dříve představeného útoku „DNS Spoofing“ v kombinaci s „ARP Poisoning“ útokem. Jejich pomocí jsme instalačnímu procesu a později i panelu podvrhli námi pozměněná data. Během instalace programu Charlie se nám podařilo skrýt instalaci druhého nástroje, který ke své instalaci potřeboval oprávnění administrátora. U panelu jsme úspěšně změnili jeho vzhled, aby jedno tlačítko obsahovalo námi vybraný obrázek a směřovalo na námi zvolenou doménu.

Ačkoliv jsme přímo v aplikaci Charlie významnou zranitelnost neobjevili, podařilo se nám objevit významnou zranitelnost v jejím instalačním programu a panelu internetového prohlížeče, který je nainstalován spolu s touto aplikací. V průběhu našeho průzkumu této aplikace vývojáři nejspíše také objevili zranitelnost v instalačním programu a vydali jeho novou verzi. Tato verze na rozdíl od testované nevyužívá síťovou komunikaci, a tím je tato významná chyba eliminována.

Literatura

[1] Cogswell, B. *AccessEnum v1.32*. 2006. Dostupné z: https://technet.microsoft.com/en-us/library/bb897332.aspx

[2] Gritzalis, D.; Kiayias, A.; Askoxylakis, I. *Cryptology and Network Security*. Heraklion, Crete: Springer, 2014, ISBN 978-3-319-12279-3.

[3] Heffner, C. *Binwalk v2.0.0.0*. 2015. Dostupné z: http://www.binwalk.org

[4] Hex-Rays. *IDA 5.0 Freeware*. 2010. Dostupné z: https://www.hex-rays.com/index.shtml

[5] LeBlance, D.; Howard, M. *Writing secure code*. Microsoft Press,U.S, 2001, ISBN 0735615888.

[6] Microsoft Corp. *XML Notepad 2007*. 2007. Dostupné z: https://www.microsoft.com/en-us/download/details.aspx?id=7973

[7] Microsoft Corp. *Dumpbin*. 2013. Dostupné z: https://support.microsoft.com/en-us/kb/177429

[8] Microsoft Corp. *Netstat*. 2015. Dostupné z: https://technet.microsoft.com/en-us/library/bb490947.aspx

[9] Microsoft Corp. *What is the AppData folder?* 2015. Dostupné z: http://windows.microsoft.com/en-us/windows-8/what-appdata-folder

[10] Microsoft Corp. *What is the registry?* 2015. Dostupné z: http://windows.microsoft.com/en-us/windows-vista/what-is-the-registry

[11] Microsoft Corp. *WinDbg v6.3*. 2015. Dostupné z: https://msdn.microsoft.com/en-us/windows/hardware/hh852365

[12] Miller, S. P. *Dependency Walker v2.2.6000*. Microsoft Corp., 2006. Dostupné z: http://dependencywalker.com/

[13] National Institute of Standards and Technology. *National Vulnerability Database*. 2015. Dostupné z: https://web.nvd.nist.gov/view/vuln/search

[14] Netmarketshare. *Operating system market share (online statistika)*. 2015. Dostupné z: http://www.netmarketshare.com/operating-system-market-share.aspx

[15] NTCore. *CFF Explorer vIII*. 2012. Dostupné z: http://www.ntcore.com/exsuite.php

[16] Oranghi, A.; Valleri, M. *Ettercap v0.8.2*. 2015. Dostupné z: http://ettercap.github.io/ettercap/

[17] Regshot. *Project Web Hosting - Open Source Software*. 2015. Dostupné z: http://regshot.sourceforge.net/

[18] Roshal, A. *WinRAR v5.21*. 2015. Dostupné z: http://www.rarlab.com/

[19] Russinovich, M. *TCPView v3.05*. 2011. Dostupné z: https://technet.microsoft.com/cs-cz/sysinternals/bb897437

[20] Russinovich, M. *Strings v2.52*. 2013. Dostupné z: https://technet.microsoft.com/cs-cz/sysinternals/bb897439.aspx

[21] Russinovich, M.; Cogswell, B. *Process Monitor v3.1*. 2014. Dostupné z: https://technet.microsoft.com/en-us/sysinternals/bb896645

[22] Seacord, R. C. *Secure Coding in C and C++*. Michigan: Addison Wesley, 2013, ISBN 978-0-321-82213-0.

[23] Shinder, D. L. *Scene of cybercrime*. Syngress Media,U.S, 2002, ISBN 1931836655.

[24] Song, D. *Dsniff v2.3*. 2015. Dostupné z: http://www.monkey.org/~dugsong/dsniff/

[25] Stevens, W. R. *UNIX Network Programming*. Prentice Hall of India, 1993, ISBN 8120307496.

[26] Weidman, G. *Penetration testing*. San Francisco, CA: No Starch Press, Inc., 2014, ISBN 1-59327-564-1.

[27] Wireshark Foundation. *Wireshark v1.12.4*. 2014. Dostupné z: https://www.wireshark.org/

[28] Yuschuk, O. *OllyDbg v2.01*. 2015. Dostupné z: `http://www.ollydbg.de/version2.html`

[29] Zahradnický, T. Running with Least Privileges, Windows User Account Control. FIT ČVUT v Praze, 2014.

[30] Zahradnický, T. Security Layers, Tokens, and ACLs. FIT ČVUT v Praze, 2014.

Příloha A

Seznam použitých zkratek

ACL Access control list

DNS Domain name system

Wi-Fi Wireless Fidelity

ARP Address resolution protocol

ASLR Address space layout randomization

DEP Data execution prevention

IP Internet protocol

HTTP Hypertext transfer protocol

HTTPS Hypertext transfer protocol secure

IT Information technology

DLL Dynamic-link library

TCP Transmission Control Protocol

UDP User Datagram Protocol

PE Portable Executable

UAC User Account Control

PID Process ID

URL Uniform resource locator

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `privilegeLevel` ..... program využitý v prvním útoku
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `img` ..... použité obrázky
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS