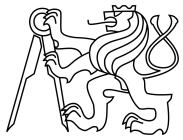
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Bezpečnostní studie aplikace
Student:	Bc. Jan Trusina
Vedoucí:	Ing. Tomáš Zahradnický, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Seznamte se s metodami a programovým vybavením používaných při zpětném inženýrství počítačového softwaru. Tyto metody a nástroje použijte k provedení bezpečnostní analýzy vedoucím zadané aplikace. Zaměřte se zejména na připojování zadané aplikace k síti Internet. Nalezené zranitelnosti zdokumentujte, vyhodnoťte a navrhněte vhodná protiopatření vedoucí k jejich nápravě.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 8. ledna 2017

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

Diplomová práce

Bezpečnostní studie aplikace

Bc. Jan Trusina

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

7. května 2017

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu, Ing. Tomáši Zahradnickému, EUR ING, Ph.D., za odborné vedení práce, cenné rady a ochotu býti nápomocen. Také bych chtěl poděkovat všem nejbližším za podporu nejen při studiu.

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 7. května 2017 .....

Odkaz na tuto práci

Trusina, Jan. *Bezpečnostní studie aplikace.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Práce se zabývá bezpečnostní analýzou běžně používané aplikace. Popisuje analýzu instalace a běhu programu, za účelem posouzení bezpečnosti aplikace. Zabývá se modelovým útokem a návrhem bezpečnostních opatření.

Klíčová slova Bezpečnostní analýza, počítačová bezpečnost, analýza programu

Abstract

This thesis deals with a security analysis of a commonly used software program. It provides an analysis of instalation and run of the program to assess it's security. It inquires into a model attack and provides security countermeausers.

Keywords Security analysis, computer security, program analysis

Obsah

Seznam obrázků

Kapitola 1

Úvod

Žijeme v digitální době. Moderní technologie se dostávají do všech odvětví, nejrůznější přístroje se připojují k internetu. Používáme běžně technologie, o kterých ještě před pár lety snili jen ti největší technologičtí vizionáři. V lékařství používáme roboty na dálku ovládané při chirurgických zákrocích, v dopravě dochází k velkému rozmachu autonomně řízených vozidel, doma si můžeme ke svému chytrému telefonu připojit rychlovarnou konvici či toustovač. Uvědomujeme si však nebezpečí naší závislosti na funkčnosti technologií?

Všechny tyto věci mají za cíl nám zpříjemnit život, ovšem jejich používání přináší i několik negativních aspektů, které si většina běžných uživatelů ani neuvědomuje. Výsledky vědeckých studií zaměřených na zkoumání vlivu elektronických zařízení na vývoj člověka se dozvíme až s odstupem času. Fakt, že používání elektronických zařízení snižuje míru našeho soukromí, už dnes ale nikoho nepřekvapuje. Nejnebezpečnějším aspektem je ovšem závislost společnosti na internetu a dalších elektronických zařízeních. Dokážete si představit, co by se stalo, kdyby týden nefungoval internet? Mohla by pošta vyplácet důchody, fungovala by státní správa a burza? Něco takového jsme mohli vyzkoušet my všichni v březnu roku 2013, kdy byly systémy různých institucí znepřístupněny pomocí DDoS útoku [15]. Co by se stalo v případě, že by někdo našel zranitelnost v programu ovládajícím autonomně řízená vozidla a převzal by nad nimi kontrolu? Myslíte, že jsou počítačové systémy ovládající vodní nádrže dostatečně zabezpečené, a co teprve systémy atomových elektráren?

Bezpečnost počítačových systémů bývá často opomíjena. Když se na nějakém projektu mají snížit výdaje, bezpečnost je často první na řadě. V poslední době se však na ni začíná klást větší důraz. Vážnost situace, kdy např. útočníci mohou ovlivnit výsledky voleb nebo způsobit havárii centrifugy při obohacování uranu [68], si začíná uvědomovat i většina představitelů států.

Největší problém zabezpečení počítačových systémů je naprostá nutnost zabezpečit každou vrstvu systému, protože síla zabezpečení celého systému je dána silou nejslabšího článku. Také musíme mít neustále na paměti, že se systémy nedají stoprocentně zabezpečit. Lze pouze minimalizovat riziko a maximalizovat úsilí, které musí útočník vynaložit na prolomení.

V tomto desetiletí pozorujeme obrovský rozvoj menších zařízení, které se připojují na internet, mluvíme o tzv. Internetu věcí (IoT). U těchto zařízení můžeme dobře vidět, jak to dopadá, když se na bezpečnost nebere ohled [31]. V minulém roce jsme mohli pozorovat následky infekce malware Mirai [22], který infikoval převážně bezpečnostní kamery a domácí routery, vytvořil z nich botnet, který následně využil k DDoS útoku. Zabezpečení

některých bezpečnostních kamer bylo tak mizivé, že americký blogger Devin Coldewey naměřil pouhých 98 vteřin, než mu malware podobný Mirai ovládl novou kameru po zapojení do sítě [10]. Za tento čas nestihne většina méně zkušených uživatelů ani najít v návodu informace o postupu změny hesla, natož následně postup realizovat.

Bezpečnost počítačových systémů má mnoho odvětví. Existuje několik metod, jak správně navrhnout a implementovat bezpečný systém, ovšem tímto se tato práce nebude zabývat. Cílem práce bude posoudit bezpečnost jedné zvolené aplikace. Při zkoumání činnosti operačního systému v aplikaci Wireshark bylo zjištěno používání síťové komunikace po nezabezpečeném kanále HTTP. Při podrobnější analýze bylo zjištěno, že se jedná o běžně používanou aplikaci a produkt významné firmy na trhu. V práci bude nutné některé klíčové detaily vynechat či začernit, aby nemohla být použita jako návod na útok na analyzovanou aplikaci. Jméno aplikace bude v práci zaměněno za symbol Kappa. Postupy uvedené v práci jsou však relativně jednoduše aplikovatelné na jakékoli programy.

Práce je členěna do šesti kapitol. Postupně se seznámíme s nástroji pro analýzu programu, tyto nástroje rozdělíme podle použití na nástroje pro analýzu vstupů programu a nástroje pro analýzu funkcionality programu. Vybereme některé z představených nástrojů a ve třetí kapitole je použijeme pro analýzu aplikace Kappa, kde budeme zkoumat nejdříve instalaci aplikace a následně její obvyklý běh. V případě nalezení bezpečnostních nedostatků v aplikaci v dalších kapitolách navrhneme model útoku. Následně jej realizujeme a navrhneme bezpečností opatření odstraňující nedostatky. Závěrem shrneme zjištěné výsledky a vyhodnotíme bezpečnost aplikace.

Kapitola 2

Programy pro analýzu

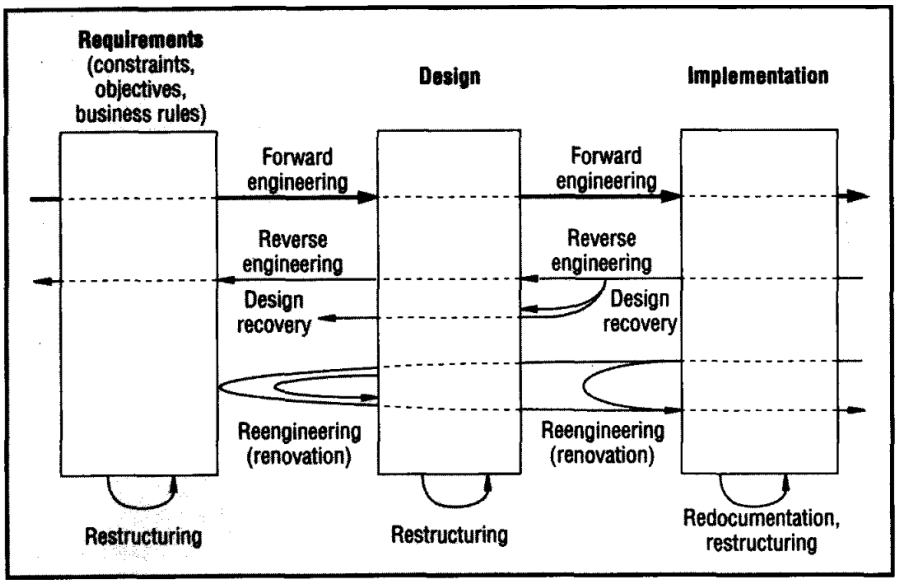
V této kapitole popíšeme použitelné nástroje pro analýzu programu Kappa. Z nich vybereme nejvhodnější nástroje, které budeme následně používat pro analýzu instalace a běhu programu s cílem posoudit jeho bezpečnost. Zkoumaný program je určený pro operační systém Microsoft Windows, proto se budeme věnovat při analýze výhradně tomuto operačnímu systému. Analýza bude vyžadovat použití mnoha nástrojů, pomocí kterých budeme zkoumat různé vlastnosti programu.

Při analýze instalátoru i samotné aplikace Kappa se zaměříme na vstupy programu. Budeme zkoumat všechny vstupní vektory, které mohou ovlivnit instalaci či obvyklý běh programu. Dále budeme zkoumat funkcionalitu aplikace a ověřovat správné použití bezpečnostních mechanismů. Toho docílíme pomocí technik reverzního inženýrství [65] a monitorovacích nástrojů. Reverzní inženýrství je proces, který jde v opačném směru proti softwarovém inženýrství. Ze zdrojového kódu vznikne překladačem kód objektový, který se sestaví nástrojem pro sestavování (linker) do spustitelné podoby ve strojovém kódu.

Při reverzním inženýrství se snažíme vzorec softwarového inženýrství aplikovat pozpátku, tedy zvyšujeme míru abstrakce pomocí procesu zvaného disassemblování [21]. Některé nástroje dokonce umožňují zpětný překlad programu, tedy překlad ze strojového jazyka do lidsky čitelného pseudokódu ve vyšším programovacím jazyce (C, Python, atd.). Reverzní inženýrství bylo definováno jako „Proces analýzy zkoumaného systému za účelem vytvoření reprezentace systému na vyšší úrovni abstrakce“, viz. obrázek 2.1.

Při reverzním inženýrství by měl být brán ohled na etický i legální aspekt tohoto konání. V České republice možnosti použití upravuje Zákon č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon). Podle tohoto zákona není povoleno používat reverzní inženýrství za účelem zkopírování a vytvoření konkurenčního programu, generátorů klíčů nebo kvůli obejití obrany zkoumaného systému. Zkoumá-li se program metodou reverzního inženýrství, využívá se statické a dynamické analýzy. Statická analýza obnáší zkoumání strojového kódu bez jeho spuštění. Dynamická analýza zkoumá strojový kód za běhu. Obě části mají odlišné výhody a nevýhody použití.

Při analýze programu se obvykle před použitím reverzního inženýrství analyzují nejdříve vstupy programu. To ulehčí následné použití reverzního inženýrství, protože už bude jasné, s jakými vstupy aplikace Kappa manipuluje. V následující sekci představíme tyto nástroje.



Obrázek 2.1: Schéma procesu zpětného inženýrství [9]

2.1 Analýzu vstupů programu

V této sekci popíšeme nástroje pro analýzu vstupů programu. Neošetřené a úmyslně špatné vstupy mohou negativně ovlivnit běh programu. Musíme předpokládat, že jakýkoli vstup může být špatný [64]. Tedy musíme ověřit, že program Kappa se všemi vstupy pracuje správně a kontroluje je. Speciálně nás bude zajímat, jak program pracuje a manipuluje s těmito vstupy:

- Data z databázových systémů
- Pojmenované roury
- Příkazová řádka
- Registrové klíče
- RPC
- Sokety
- Soubory

Tato odvětví pro analýzu vstupů programu jsme definovali pro přehlednost, ale vybrané nástroje velmi často využijeme k analýze více typů vstupů současně. Jelikož nebylo nalezeno žádné použití databáze, pojmenovaných rour, příkazové řádky ani RPC, nebudeme se těmto vstupům dále věnovat.

2.1.1 Bezpečnostní objekty

Nejprve představíme programy pro analýzu souborů a registrových klíčů. Tyto dvě položky z předešlého seznamu jsme nyní sloučili, protože mají jednu společnou věc - operační systém

Microsoft Windows je považuje za bezpečnostní objekty (SO). Musíme tedy nejdříve definovat, jak má operační systém Microsoft Windows rozvrstvené zabezpečení [60]. S trochou nadsázky můžeme konstatovat, že zabezpečení operačního systému Windows je o zkratkách [66]. Tyto pojmy běžně používané v Anglickém jazyce nebudeme překládat, abychom jejich porozumění příliš nekomplikovali.

- Trustee — označuje uživatele, skupinu uživatelů či sezení
- Security Identifier (SID) — unikátní identifikátor Trustee
- Access Control Entry (ACE) — identifikovanému Trustee přiřazuje přístupová práva:
  - Granted — potvrzena;
  - Denied — zamítnuta
  - Audited — systémem revidována
- Access Control List (ACL) — seznam ACE
- Securable Object (SO) — zabezpečené objekty, které mají Security Descriptor:
  - IPC objekty
  - Pojmenované a nepojmenované roury
  - Procesy a vlákna
  - Přístupové tokeny
  - Registrové klíče
  - Sdílené disky, tiskárny atd.
  - Soubory a složky na souborovém systému NTFS; FAT32 toto zabezpečení nepodporuje
- Discretionary Access Control List (DACL) — seznam ACE, který potvrzuje či zamítá přístup k SO
- System Access Control List (SACL) — seznam ACE, jejichž přístup k SO je auditován
- Security Descriptor (SD) — popisuje bezpečnostní informace SO, skládá se ze SID vlastníka, SID skupiny vlastníka, DACL, SACL a příznaků

Představili jsme základní pojmy v zabezpečení, které používá operační systém Microsoft Windows. Nyní popíšeme nástroje pro analýzu vstupů registrových klíčů a souborů.

2.1.2 Analýza manipulace s registrovými klíči

Registr Windows byl poprvé použit v operačním systému Microsoft Windows 3.11 v roce 1992, kdy rozšiřoval nedostatečné možnosti konfiguračních INI souborů. Jedná se o hierarchický systém pro ukládání klíčů a jejich hodnot sloužících k nastavení jádra, ovladačů, služeb operačního systému či uživatelských programů. Z bezpečnostního hlediska předpokládáme, že veškerá funkcionalita operačního systému je v pořádku.

Budou nás zajímat všechny registrové klíče, které aplikace Kappa čte, a které mají slabé ACL. To mohou být například COM objekty [6], které definují COM servery, uložené v DLL se spustitelným kódem. Identifikátory COM objektů se ukládají do registrových klíčů v lokaci `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\`. Pokud by bylo možné změnit klíč s registrací COM objektu a nebo jeho DLL, bylo by možné spustit kód. Pro analýzu práce s registrovými klíči můžeme použít následující programy:

- Regshot
- Process Monitor
- Autoruns
- RegFromApp
- Programy operačního systému

2.1.2.1 Regshot

Tento program[1] s otevřeným zdrojovým kódem je dostupný pod licencí LGPL. Slouží k vytvoření dvou otisků současného stavu registrových klíčů a jejich následnému porovnání. Kromě registrových klíčů zde můžeme nadefinovat i sledování adresářů na souborovém systému. Program po porovnání vypíše všechny registrové klíče, které byly přidány, upraveny či odebrány. Regshot nesleduje jednotlivé procesy, ale pouze změny v registrových klíčích, tudíž zaznamenává všechny změny všech procesů. Výhodou tohoto programu je, že může být během běhu sledovaného programu vypnutý. To bychom ocenili například při sledování potenciálně nebezpečného programu, který by maskoval své chování při detekci nějakého sledovacího či ladícího nástroje.

2.1.2.2 Process Monitor

Tento program, zkráceně ProcMon, je z balíčku Sysinternals Suite [48], který vytvořil Mark Russinovich v roce 1996. O deset let později tento celý produkt akvizoval Microsoft Corp. a Mark Russinovich se stal vedoucím v Microsoft Azure. Balíček tvoří sada 70 nástrojů, které usnadňují analýzu, diagnostiku a hledání chyb v programech pro operační systémy Microsoft Windows.

ProcMon na rozdíl od programu Regshot zaznamenává veškeré operace [4] s registrovými klíči, a to díky tomu, že program běží spolu se sledovaným procesem a živě zaznamenává informace o činnosti sledovaného procesu. Kromě registrových klíčů sleduje také síťovou aktivitu, činnost jader procesorů a vláken, profilování a činnost souborového systému. Tento monitorovací program umožňuje filtrování událostí podle mnoha parametrů (jméno procesu, operace, času, PID, atd.) [27], což je nutné vzhledem k vysokému počtu instrukcí, které se provádějí. Díky možnosti filtrování a zaznamenávání vývoje v reálném čase se jedná o vhodný doplněk k předešlému nástroji Regshot.

[1] Odkaz na webové stránky aplikace Regshot: `https://sourceforge.net/projects/regshot/`

2.1.2.3 Autoruns

Tato aplikace, další z dílny Marka Russinoviche a jeho balíčku Sysinternals [48], sleduje všechny procesy, které se spouštějí při startu počítače. Tato nastavení se ukládají do registrových klíčů, tudíž tento nástroj použijeme k prohlížení zde uložených hodnot souvisejících se startem počítače (Autostart Extensibility Points). Aplikace umožňuje také ověřit digitální certifikát spouštěných souborů a případně je nechat zkontrolovat přímo službou VirusTotal[2]. Jelikož některé škodlivé programy skrývají své registrové klíče v napadeném operačním systému, je možné vytvořit virtuální kopii a tu následně pomocí aplikace Autoruns offline prověřit. Samozřejmostí je vygenerování záznamu událostí a uložení do formátu CSV pro usnadnění importu do dalších nástrojů.

2.1.2.4 RegFromApp

Tento nástroj[3] s otevřeným zdrojovým kódem slouží k zobrazení změn v registrových klíčích. Po spuštění nástroje můžeme vybrat sledovanou aplikaci, nástroj poté začne monitorovat chování aplikace a následně zobrazí změny v registrech včetně původních hodnot, které provedla sledovaná aplikace.

2.1.2.5 Programy operačního systému

V případě, že bychom chtěli sledovat změny registrových klíčů bez použití softwaru třetích stran, můžeme použít programy, které jsou již nainstalované a připravené operačním systémem. Pomocí programu Regedit (Editor registru Windows) vyexportujeme, obdobně jako při použití programu Regshot, snímek registrových klíčů před a po instalaci. Následně snímky porovnáme programem Fc (File compare).

2.1.3 Analýza manipulace se soubory

Soubory se v operačním systému Microsoft Windows považují za bezpečnostní objekty (viz 2.1.1). Při analýze manipulace se soubory nás bude zajímat, se kterými soubory se manipuluje, a zda jsou správně zabezpečeny. Při analýze budeme primárně vycházet z již analyzovaných programů Regshot (viz 2.1.2.1) a Process Monitor (viz 2.1.2.2). Dále zde popíšeme programy AccessEnum, AccessChk a DiskPulse.

2.1.3.1 AccessEnum a AccessChk

Tyto nástroje jsou součástí balíčku nástrojů na správu od SysInternals [48]. AccessEnum zobrazuje a umožňuje měnit přístupová práva k souborům, složkám a registrovým klíčům. Funkcionalitou je tento nástroj podobný dalšímu nástroji z balíčku AccessChk, který se ovládá z příkazové řádky a umožňuje zkoumat veškeré bezpečnostní objekty operačního systému Microsoft Windows.

[2]Odkaz na webovou stránku `https://virustotal.com/`

[3] Odkaz na webové stránky aplikace RegFromApp `http://www.nirsoft.net/utils/reg_file_from_application.html`

2.1.3.2 DiskPulse

Komerční nástroj DiskPulse[4] se používá pro sledování změn na celém disku. Nástroj umožňuje filtrovat, sledovat a analyzovat změny na disku v reálném čase. V případě dlouhodobějšího zaznamenávání aktivity je možné připojit aplikaci k SQL databázi, kam se zaznamenávají výsledky. Tento typ aplikace bychom pro analýzu zvolili v případě, že by sledovaná aplikace Kappa často manipulovala se soubory na různých discích.

2.1.4 Analýza síťové komunikace

Důkladná analýza síťové komunikace je stěžejní část každé bezpečnostní studie, protože tato komunikace obsahuje mnoho potenciálních vektorů útoku. Jak již bylo řečeno v úvodu, bezpečnost aplikace se určuje podle bezpečnosti nejslabšího článku, a dost často tento nejslabší článek bývá v síťové komunikaci. Musíme ověřit, zda aplikace komunikuje po síti. Pokud komunikuje, tak zda používá šifrované spojení se servery, zda jsou správně používány a ověřovány certifikáty. K analýze síťové komunikace můžeme použít programy Wireshark, Process Monitor a TCPView. Process Monitor jsme popsali již dříve, tudíž se s ním znovu nebudeme zabývat.

2.1.4.1 Wireshark

Tento program [8] napsal v roce 1998 Gerald Combs, jedná se o další software s otevřeným zdrojovým kódem dostupným pod licencí GPL. Wireshark se používá pro analýzu síťových paketů či komunikace USB. Program je multiplatformní, umožňuje offline analýzu provozu a podrobné zkoumání mnoha protokolů. Tento program je připojen přímo na monitorované rozhraní, tudíž není možné filtrovat provoz např. podle jména procesu. Z toho důvodu bude pro naši práci vhodné zkombinovat výstupy z aplikace Wireshark spolu s výstupy z aplikace Process Monitor, abychom mohli přesně analyzovat chování monitorované aplikace.

2.1.4.2 TCPView

Tento nástroj od SysInternals [48] umožňuje v reálném čase sledovat seznam všech koncových bodů spojení (endpoint) na transportní vrstvě OSI modelu. Jednotlivé položky jsou tříděny podle názvu procesu, který je vytvořil. Dále program zobrazuje informace o PID procesu, použití protokolu na transportní vrstvě (TCP, UDP), verzi IP protokolu (IPv4, IPv6), portech, lokální a vzdálené adrese. U protokolu TCP navíc zobrazuje i stav, ve kterém se spojení nachází.

[4] Odkaz na webové stránky aplikace DiskPulse `http://www.diskpulse.com/index.html`

2.2 Nástroje pro analýzu funkcí programu

V první části kapitoly jsme rozdělili nástroje podle využití pro analýzu vstupů a analýzu funkcí programu. První část jsme již popsali stejně tak jako nástroje, které v rámci její analýzy můžeme využít. Zbývá popsat druhou část, nástroje pro analýzu funkcí programu. Základní dělení použitých nástrojů zde bude podle využití:

- Zjišťování informací z binárních souborů
- Reverzní inženýrství
- Sledování chování procesu

Toto rozdělení nástrojů odpovídá i obvyklému pracovnímu postupu při analýze funkcí programu. Nejdříve je vhodné zjistit z binárních souborů informace o použitých knihovnách, překladači, použití kompresního nástroje atd., a až poté se pustit do reverzního inženýrství.

2.2.1 Zjišťování informací z binárního souboru

Při analýze binárních souborů budeme hledat bezpečnostní opatření programu a zkoumat, zda jsou implementována korektně. Dále budeme analyzovat, zda použité staticky i dynamicky linkované knihovny jsou bezpečné, zda importované knihovny i program samotný používají zabezpečení ASLR (Address Space Layout Randomization), DEP (Data Execution Prevention) a bezpečný SEH (Structured Exception Handling) [50], či zda jsou binární soubory zabalené pomocí nějakého kompresního nástroje [49]. Vhodné nástroje pro výše uvedené jsou zejména:

- CFF Explorer
- PE Explorer
- ExeInfoPE
- Resource Hacker
- Strings

2.2.1.1 CFF Explorer

Tento nástroj [43] vytvořil Daniel Pistelli, aby sobě i následně dalším uživatelům ulehčil čtení a analýzu Portable Executable souborů [42]. Jedná se o volně dostupnou verzi komerčního produktu firmy Cerbero Profiler[5]. PE formát souboru zahrnuje klasické spustitelné soubory, knihovny, objektové a systémové soubory a několik dalších typů souborů. Nástroj CFF Explorer můžeme použít pro analýzu programu Kappa, abychom snadno získali čitelné informace o hlavičce a jejích sekcích, abychom zjistili, zda se jedná o program 64bitový či 32bitový, dále kvůli ověření používání bezpečnostních mechanismů ASLR, DEP nebo bezpečného SEH. Program má v sobě zabudovanou také stejnou funkcionalitu na zkoumání dynamicky linkovaných knihoven, kterou poskytuje nástroj Dependency Walker [53].

[5] Odkaz na webové stránky firmy Cerbero Profiler `http://cerbero.io/profiler/`

2.2.1.2 PE Explorer

Komerční nástroj[6] od společnosti Heaventools Software pro 32bitové aplikace umožňuje zkoumat a měnit prvky grafického rozhraní, analyzovat vlastnosti PE souboru, sledovat přístupy programu a volané knihovny. Dále PE Explorer umožňuje ověřovat podpisy souborů a rozbalovat zabalené programy. Software také nabízí nástroje pro analýzu programů napsaných v jazyce Delphi.

2.2.1.3 ExeInfoPE

Tato aplikace[7] s otevřeným zdrojovým kódem při analýze binárního souboru slouží k identifikaci překladače, architektury, nástroje na sestavování programu či informace, zda byl soubor zabalen pomocí kompresního nástroje [49]. Nástroj si poradí i s většinou souborů chráněných proti takovéto analýze.

2.2.1.4 Resource Hacker

Další nástroj[8] s otevřeným zdrojovým kódem slouží ke zkoumání a případným úpravám zdrojů 32 i 64bitových spustitelných souborů pro operační systém Microsoft Windows. Umožňuje úpravu a překlad .rsrc sekce PE souborů. Analýzou pomocí tohoto programu zjistíme, zda program neobsahuje nějaké další vnořené programy či soubory (např. XML). Dále ho budeme využívat kvůli snadnému čtení či další manipulaci s manifestem aplikace.

2.2.1.5 Strings

Další nástroj z dílny Marka Russinoviche a jeho balíčku SysInternals [48] vyextrahuje z PE souboru cokoli, co připomíná textový řetězec. Staticky linkované knihovny se hledají velice obtížně, tudíž můžeme doufat, že v seznamu textových řetězců bude uložený nějaký řetězec obsahující copyright, verzi, název produktu či podobnou informaci, pomocí které bychom mohli identifikovat, o jakou knihovnu se jedná.

2.2.2 Nástroje na reverzní inženýrství

V této části popíšeme nástroje, které použijeme při analýze funkcionality programu Kappa. K tomu využijeme statickou a dynamickou analýzu programu. Nástroje na reverzní inženýrství budeme moci použít tyto:

- Binary Ninja
- IDA
- Immunity Debugger
- OllyDbg
- WinDbg
- x64dbg

[6] Odkaz na webové stránky aplikace PE Explorer `http://www.heaventools.com/overview.htm`
[7] Odkaz na webové stránky aplikace ExeinfoPE `http://exeinfo.pe.hu/`
[8] Odkaz na webové stránky aplikace Resource Hacker: `http://www.angusj.com/resourcehacker/`

2.2.2.1 Binary Ninja

Nově vzniklý komerční nástroj[9] od společnosti Vector 35 LLC byl vydán v první verzi v létě roku 2016. Nástroj zatím umožňuje pouze statickou analýzu pomocí disassemblování, ale na rozdíl od většiny ostatních nástrojů umí zobrazit instrukce do přehledného grafu toku. V plánu je rozšíření nástroje o nástroje na zpětný překlad a ladění či možnost umístit zkoumaný soubor do odizolovaného prostředí (sandbox).

2.2.2.2 IDA

Tento komerční nástroj [20], v překladu Interaktivní disassembler, se používá pro zpětný překlad, ladění a disassemblování. Díky přehlednému grafu toku instrukcí je vynikajícím nástrojem pro statickou i dynamickou analýzu. Je volně dostupný ve staré verzi IDA Pro 5.0 Free[10] pro nekomerční účely, ovšem pouze bez zpětného překladače a bez podpory 64bitových aplikací.

2.2.2.3 Immunity Debugger

Tento ladící nástroj[11] přišel jako první na trh s možností analýzy haldy. Nástroj umožňuje snadnou rozšiřitelnost pomocí vlastních zásuvných modulů skriptovaných v jazyce Python. Nástroj je možné připojit k dalším nástrojům na fuzzing (testování vstupů programu) či nástrojům pro ulehčení vývoje exploitů. Immunity Debugger je možné ovládat přes přehledné grafické rozhraní, nebo pomocí příkazové řádky, což může ulehčit přechod z podobných ladících nástrojů (například WinDbg) i díky používání aliasů.

2.2.2.4 OllyDbg

Tato ladící aplikace [16] je dostupná pod licencí GPLv3. Jedná se o klasický ladící nástroj pro 32bitové programy, který se ovládá přes grafické rozhraní. Pro ladění 64bitových aplikací je možné použít alfa verzi OllyDbg 2.01. Nástroj umožňuje nastavovat body přerušení, kontrolovat hodnoty uložené v registrech a pod.

2.2.2.5 WinDbg

Oficiální ladící nástroj[12] společnosti Microsoft Corp. je určený pro ladění 32 i 64bitových aplikací a jádra operačního systému Microsoft Windows. WinDbg se obvykle ovládá pomocí příkazové řádky a můžeme jej využít například při hledání příčiny pádu jádra operačního systému. Při ladění jádra operačního systému, který běží na virtuálním stroji, se používají pojmenované roury a virtuální COM porty. Nástroj je oficiálně komerční, ovšem pro majitele operačního systému Microsoft Windows je volně dostupný v rámci balíku: Debugging Tools for Windows.

[9] Odkaz na webové stránky aplikace Binary Ninja `https://binary.ninja/`

[10] Odkaz na webové stránky aplikace IDA PRO Free 5.0 `https://www.hex-rays.com/products/ida/support/download_freeware.shtml`

[11] Odkaz na webové stránky aplikace Immunity Debugger `https://www.immunityinc.com/products/debugger/index.html`

[12] Odkaz na webové stránky aplikace WinDbg `http://www.microsoft.com/whdc/devtools/debugging/default.mspx`

2.2.2.6 x64dbg

Tuto aplikaci[13] na ladění můžeme použít na rozdíl od OllyDbg při analýze 64bitových aplikací. Jedná se o poměrně nový program dostupný pod licencí GPLv3. Aplikace je dostupná s otevřeným zdrojovým kódem, snadno rozšiřitelná a například zásuvný modul Scylla umí šikovně vypisovat procesy z paměti [62].

2.2.3 Nástroje na sledování chování procesu

V rámci analýzy funkcionality programu bude vhodné použít nástroje na sledování chování programu. Již byly popsány podrobně používané vstupy programu a nástroje na jejich sledování. Dále jsme popsali nástroje na reverzní inženýrství a zjištění informací z binárních souborů. Zbývá tedy představení nástrojů na sledování volaných API a kontrolu bezpečnostního tokenu. V rámci bezpečnostní analýzy aplikace je nutné ověřit, že aplikace Kappa ctí princip minimálních práv.

Operačním systému Microsoft Windows popisuje bezpečnostní vlastnosti (Security Context) pomocí souboru aktuálních bezpečnostních nastavení a pravidel. Přístupový token je objekt, který pomocí tohoto souboru popisuje bezpečnost procesu nebo vlákna [67]. Na operačních systémech Microsoft Windows Vista a novější se po úspěšném přihlášení vytvoří primární token. Každý proces má vlastní bezpečnostní token, tudíž při spuštění nového procesu se vytváří nový bezpečnostní token, který může mít oproti tokenu rodičovského procesu stejně či méně práv.

2.2.3.1 TokenMaster

Aplikace TokenMaster [45] byla představena v rámci předmětu MI-BPR. Aplikace zkoumá bezpečnostní token procesu a zobrazí informace o jeho vlastnostech. Nástroj také umožňuje spustit nový proces s vlastnostmi načteného tokenu.

2.2.3.2 API Monitor

Tento nástroj [2] s otevřeným zdrojovým kódem umožňuje sledovat volání API programu, v přehledném grafickém rozhraní lze pozorovat podrobně chování aplikace včetně volaných funkcí.

[13] Odkaz na webové stránky aplikace x64dbg `http://x64dbg.com`

2.3 Výběr nástrojů

Představili jsme mnoho programů, které bychom mohli využít pro analýzu aplikace Kappa za účelem vyhodnocení její bezpečnosti. Z finančních důvodů není možné využít komerční nástroje, ačkoli např. IDA Pro by se jistě velmi hodila. Pro analýzu manipulace s registrovými klíči použijeme nástroje Regshot, Process Monitor a Autoruns. Pro analýzu manipulace se soubory využijeme kromě nástrojů Regshot a Process Monitor také nástroj AccessEnum. V rámci analýzy síťové komunikace použijeme dále aplikaci Wireshark. Pro zjišťování informací z binárního souborů využijeme nástroje CFF Explorer, ExeInfoPE, Resource Hacker a Strings. Dále v rámci analýzy funkcí programu použijeme ladící nástroj x64dbg, TokenMaster pro kontrolu bezpečnostního tokenu a API Monitor pro sledování volaných API.

2.4 Shrnutí

V této kapitole jsme představili možné nástroje pro analýzu vstupů a funkcí programu Kappa. Definovali jsme bezpečnostní objekty tak, jak je operační systém Windows používá. Nástroje pro analýzu vstupů jsme rozdělili podle jejich využití na nástroje zkoumající práci s registrovými klíči, manipulaci se soubory a síťovou komunikaci. Dále jsme popsali nástroje pro analýzu funkcí programu, které jsme rozdělili podle zaměření na nástroje zabývající se získáním informací z binárních souborů a na nástroje na reverzní inženýrství, popsali jsme rozdíl mezi statickou a dynamickou analýzu. V závěru kapitoly jsme vybrali z představených aplikací ty, které dále využijeme pro analýzu aplikace Kappa v příští kapitole, kde budeme analyzovat instalaci a obvyklý běh programu Kappa.

Kapitola 3

Analýza aplikace

V předešlé kapitole jsme popsali nabídku nástrojů pro analýzu programu Kappa, některé z nich v této kapitole použijeme. Budeme se zabývat nejdříve analýzou instalace programu, poté analýzou běhu programu. Cílem této analýzy bude posoudit bezpečnost aplikace. Zkoumáním správnosti použití bezpečnostních mechanismů zjistíme, zda byla aplikace implementována korektně podle schémat bezpečného programování [28]. Při odhalení nedostatku v zabezpečení se pokusíme vyhodnotit jeho závažnost ohodnocením rizika.

3.1 Výběr aplikace

Analyzovaná aplikace je běžně používaná napříč operačními systémy Microsoft Windows, distribucemi Linux a macOS. Při sledování chování počítače s operačním systémem Microsoft Windows 10 byl objeven proces, který komunikuje se serverem po nezabezpečeném kanálu HTTP [25]. Po diskuzi s vedoucím bylo rozhodnuto, že se bezpečnosti aplikace bude autor věnovat v rámci této diplomové práce.

Tato práce nemá sloužit jako návod k útoku na aplikaci, z toho důvodu budou začerněny veškeré informace, které by vedly k zjištění původu aplikace. Délka začerněného bloku bude vždy 5 znaků a původní informace nebude obsažena ani ve zdrojovém textu této práce. Jméno aplikace a majitelů produktu bude vždy nahrazeno symbolem Kappa.

3.2 Analýza procesu instalace programu

V této části budeme analyzovat instalaci aplikace Kappa. Průběh budeme zaznamenávat a analyzovat již od počátku, začneme tedy stažením instalátoru z internetových stránek. Při analýze instalace se budeme zabývat vstupy aplikace, budeme například ověřovat, zda jsou všechny důležité instalované komponenty chráněny dostatečným ACL [67] apod.

3.2.1 Stažení instalátoru

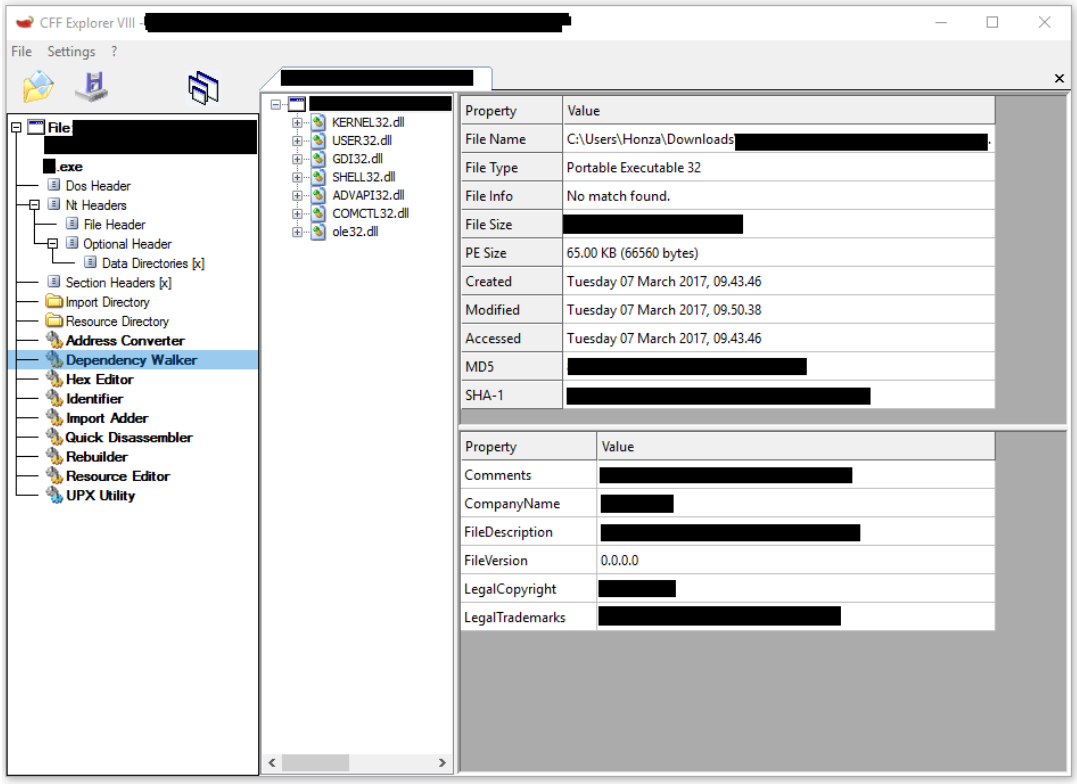
Spustíme internetový prohlížeč a ze stránek výrobce stáhneme instalátor. Komunikace s webovým serverem probíhá po nezabezpečeném kanále HTTP. Stránky umožňují zabezpečenou komunikaci pomocí protokolu HTTPS [60], ovšem automaticky k přesměrování na zabezpečený kanál nedochází. Pro stažení instalátoru máme na internetových stránkách na výběr ze dvou možností. Můžeme stáhnout instalátor obsahující všechna potřebná

data nebo instalátor minimalizovaný, který stáhne všechna potřebná data až při instalaci. Stránka nabízí implicitně celý instalátor, tudíž jej stáhneme. Instalátor se stahuje z jiné domény, nyní pomocí zabezpečeného protokolu: https://www2.[redacted].com/[redacted].exe.

3.2.2 Analýza instalačního programu

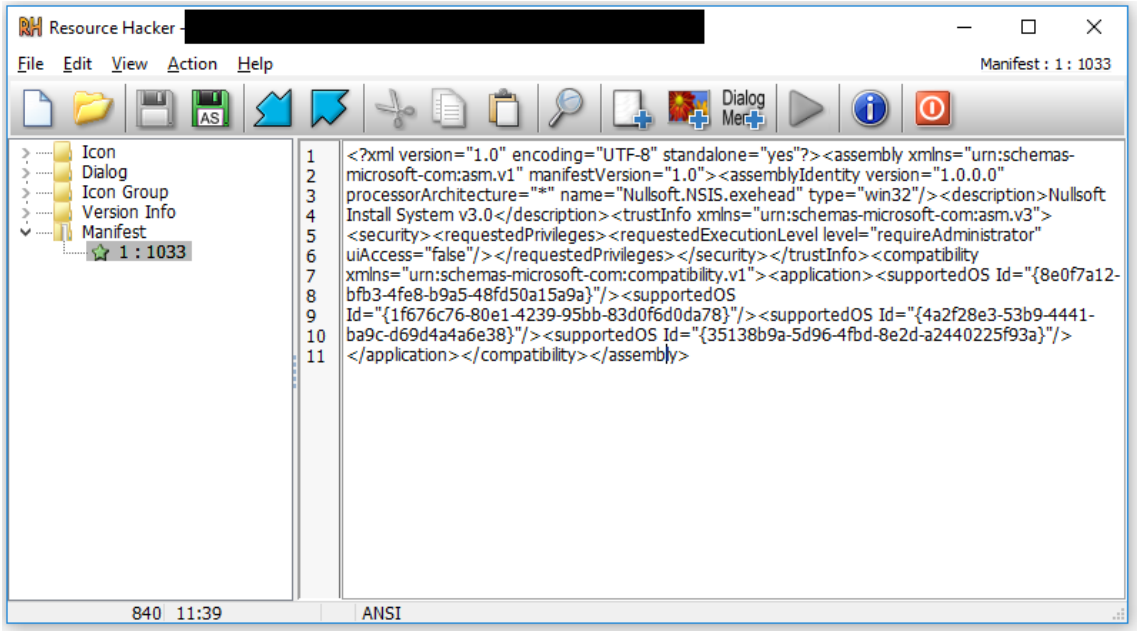
Stažený soubor je relativně velký, z toho důvodu můžeme očekávat, že se jedná o samorozbalovací archív. Informaci můžeme ověřit programem ExeInfoPE (2.2.1.3), jenže soubor je příliš velký, a tudíž tento nástroj žádné informace nezobrazí. Soubor otevřeme v nějaké starší verzi programu 7zip[14], protože nejnovější verze 15 má s otevíráním samorozbalovacích archívů problémy. Podařilo se otevřít spustitelný soubor v programu 7zip a vidíme, že se skutečně jedná o samorozbalovací archív.

Pomocí programu CFF Explorer (2.2.1.1) zjišťujeme, že instalátor je architektury x86, používá DEP (Data Execution Prevention) a ASLR (Address Space Layout Randomization), nepoužívá bezpečný SEH (Structured Exception Handling) [50]. Dále zjišťujeme (viz obrázek 3.1), že instalátor dynamicky spojuje několik běžných knihoven, které mají příznak v `DLL-Charakteristics` [42] části hlavičky `Optional Header` hodnotu 4140. Tyto knihovny používají ASLR, DEP i bezpečný SEH. Dále použijeme program Resource Hacker (2.2.1.4), pomocí kterého zkoumáme manifest souboru. Nalézáme parametr `requestedExecutionLevel level="requireAdministrator"`, který definuje, že aplikace bude i pro samotné rozbalení archívu vyžadovat administrátorský práva (viz obrázek 3.2).



Obrázek 3.1: Analýza knihoven instalátoru programem CFF Explorer

[14]Odkaz na stránky aplikace 7zip: `http://www.7-zip.org/download.html`



Obrázek 3.2: Analýza manifestu instalátoru programem Resource Hacker

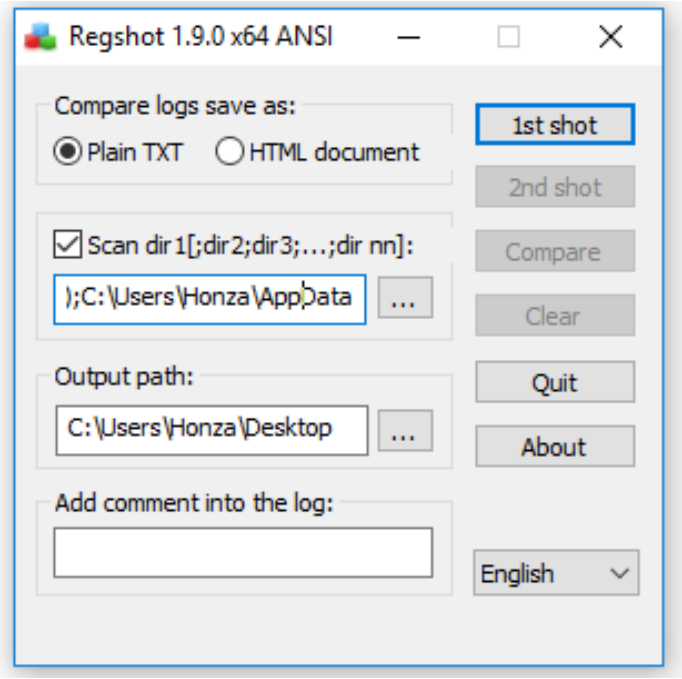
Před spuštěním instalátoru zkontrolujeme certifikát. Program je podepsaný správně vydavatelem, certifikát je platný až do roku 2019 a k dispozici jsou pro ověření dva podpisy: SHA1 a SHA256 [35]. Vhodnost použití SHA1 může být diskutabilní vzhledem k nedávno nalezené kolizi v této hašovací funkci [55]. Při změně bytu v souboru dojde k zneplatnění podpisu, což se projeví při spuštění instalátoru v okně UAC při elevaci, kde místo vydavatele Kappa je napsáno `Vydavatel: Neznámý`. Instalace se však spustí a proběhne bez problému. Aplikace Kappa tedy neověřuje svůj digitální podpis [24].

3.2.3 Instalace programu Kappa

Nejprve vytvoříme snímek registrových klíčů a souborového systému pomocí programu Regshot (2.1.2.1). Program snímá všechny registrové klíče a námi specifikované adresáře. Vytvoříme dva snímky před a po instalaci, program snímky porovná a vypíše všechny změny. Budeme sledovat tyto adresáře:

```
/Počítač
├── %USERPROFILE%\AppData
├── C:\Kappa
├── C:\Program Files
├── C:\Program Files (x86)
└── C:\Windows
```

Na obrázku 3.3 můžeme vidět snímek běhu programu Regshot. V tomto programu vybereme adresáře, jejichž snímek si program vytvoří, a složku pro výstup. Program spustíme a uložíme snímek adresářů a registrových klíčů před instalací. Dále spustíme program Proces Monitor (2.1.2.2), kterým budeme monitorovat celý průběh instalace. V tomto programu nastavíme filtrování podle jména procesu instalátoru.



Obrázek 3.3: Nastavení běhu programu Regshot

Spustíme samorozbalovací archív, který požaduje potvrzení elevace práv v okně UAC. Archív byl vytvořený pomocí programu NSIS [47], jak jsme si mohli všimnout již v manifestu (3.2). Program se rozbalí implicitně do složky, kterou jsme popsali v úvodu této sekce. Tato složka má pro běžného uživatele přístupová práva nastavená na čtení a spouštění pro všechny uživatele [17], ale právo zápisu vyžaduje vyšší přístupová práva. Složky má tedy správně nastavený ACL [67].

Po rozbalení se spustí automaticky instalační program [redacted].exe, který nainstaluje veškeré potřebné komponenty. Tento proces běží se zděděnými [54] administrátorskými právy svého rodiče — samorozbalovacího programu. Po dokončení instalace uložíme log z aplikace Process Monitor a restartujeme počítač. Poté vytvoříme druhý snímek pomocí programu Regshot a porovnáme jej s prvním snímkem.

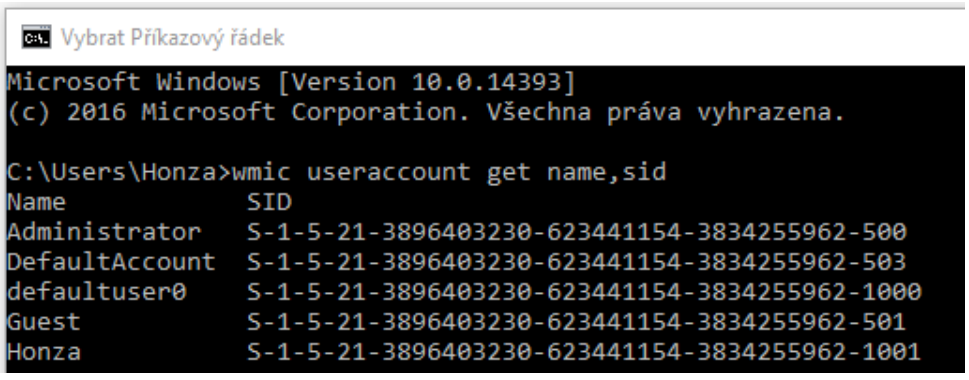
Dále budeme analyzovat zaznamenanou činnost instalátoru a vyhodnotíme, zda jednotlivé operace byly bezpečné. Budeme se nejdříve zabývat analýzou zápisu registrových klíčů, dále pak analýzou zápisu souborů. Nakonec v rámci této části budeme analyzovat síťovou aktivitu.

3.2.4 Analýza registrových klíčů

Programem Regshot (2.1.2.1) jsme vygenerovali soubor, kde jsou vypsané všechny registrové klíče a soubory, ve kterých došlo ke změně nebo byly přidané. Musíme ověřit v logu z aplikace Process Monitor (2.1.2.2), že změny byly způsobené skutečně procesem instalátoru. Změn v registrových klíčích zaregistroval Regshot několik tisíc. Z logu vyplývá, že došlo během instalace ke změnám ve větvích HKLM[15], HKU[16], HKCR[17] a HKCU[18]. Graficky zobrazené větve, do kterých bylo zapisováno tedy vypadají takto:

```
/Počítač
├── HKLM
│   ├── Kappa
│   ├── SOFTWARE
│   │   ├── Classes
│   │   │   ├── CLSID
│   │   │   └── Directory\ShellEx\ContextMenuHandlers
│   │   ├── Kappa
│   │   ├── Microsoft\Windows\CurrentVersion
│   │   └── WOW6432Node
│   └── SYSTEM\CurrentControlSet\Services
└── HKU
    ├── .Default\SOFTWARE
    └── S-1-5-21-3896403230-623441154-3834255962-1001\SOFTWARE
```

Zjistili jsme, že dochází k obvyklým zápisům do registrových klíčů přímo ve větvích HKLM a HKU. Nejdůležitějším zjištěním je zápis do Classes\CLSID, kam se ukládají záznamy registrovaných COM objektů [6]. Při zkoumání klíčů a hodnot zapsaného COM objektu zjišťujeme, že klíč `InProcServer32` odkazuje na knihovnu umístěnou v podadresáři Program Files. Tento adresář má omezená přístupová práva, tudíž budeme považovat použití COM objektů za bezpečné. Dále pozorujeme registraci klíčů ovlivňující kontextové menu, což můžeme ověřit sami otevřením kontextového menu. Ve větvi HKU došlo k zápisu do SID `S-1-5-21-3896403230-623441154-3834255962-1001`, které odpovídá současnému uživateli (viz obrázek 3.4).



Obrázek 3.4: Snímek příkazu pro zjištění uživatele a jeho SID

[15] Zkratka pro větev HKEY_LOCAL_MACHINE
[16] Zkratka pro větev HKEY_USERS
[17] Zkratka pro větev HKEY_CLASSES_ROOT, odkaz na HKLM\Software\Classes
[18] Zkratka pro větev HKEY_CURRENT_USER, jedná se o podvětev HKU

Zaznamenaných změn v registrových klíčích bylo několik tisíc, zde jsme zmínili pouze pro analýzu stěžejní větve. Analýza však neodhalila žádné bezpečnostní pochybení v činnosti instalace s Registrem Windows.

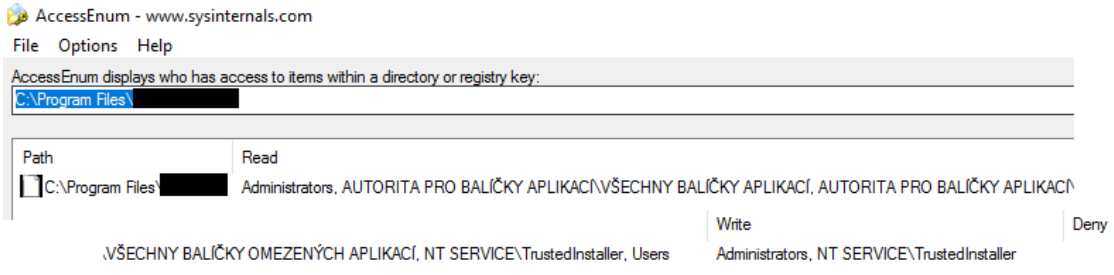
3.2.5 Analýza souborů

Po analýze změn v registrových klíčích přichází na řadu analýza zapisovaných souborů. Všechna potenciální místa pro zápis jsme označili programem Regshot. Následnou analýzou výsledků spolu s hledáním v logu aplikace Process Monitor zjistíme, že došlo k zápisu do adresářů:

```
C:\
├── %USERPROFILE%\AppData\Local\Temp
├── Kappa
├── Program_Files\Kappa
├── Program_Files_(x86)\Kappa
└── Windows
    ├── Installer
    ├── Prefetch
    └── System32
```

Ze struktury výše je zřejmé, že aplikace používá standardní adresáře. Musíme ověřit, že aplikace nemá vstupy v některé ze složek s mírnějšími přístupovými omezeními. V případě, že by běžný uživatel mohl měnit binární nebo konfigurační soubory, které mohou změnit běh programu, jednalo by se o bezpečnostní nedostatek. Jediná složka se sníženými přístupovými právy pro běžného uživatele je složka TEMP v AppData. Do této složky se ukládají pouze logy s informacemi o průběhu instalace. Můžeme tedy konstatovat, že aplikace pracuje se soubory v souladu s bezpečnostními pravidly [26].

Dále musíme zkontrolovat, zda v průběhu instalace nedošlo ke změně přístupových práv libovolné složky, která obsahuje spustitelné soubory. Toto bohužel často bývá bezpečnostním pochybením, kdy autoři aplikace přidávají pro běžného uživatele přístupová práva k souborům v Program Files, ačkoli k tomu stačí pouze, aby bezpečnostní token obsahoval právo `SeChangeNotifyPrivilege` [36]. Žádná úprava přístupových práv do složek s omezeným přístupem však nebyla zaznamenána. Nastavení jednotlivých adresářů můžeme zkontrolovat ručně, pomocí programu AccessEnum (2.1.3.1), či v aplikaci Process Monitor využitím volby `File summary`, která obsahuje sloupec `Set ACL`. Vše se zdá být v pořádku, nikde jsme neodhalili změnu ACL. Můžeme tedy konstatovat, že se aplikace chová v souladu s bezpečnostními standardy z hlediska práce se soubory.



Obrázek 3.5: Zkoumání přístupových práv do složky programem AccessEnum

3.2.6 Analýza síťové komunikace

Analýza síťové komunikace nejdříve prozradila, jakým způsobem byl instalátor stažen z internetu. Jelikož vše potřebné bylo staženo v instalátoru, můžeme předpokládat, že instalátor již příliš aktivní v síťové komunikaci nebude. Z logu aplikace Process Monitor zjistíme, že instalátor komunikoval pouze se servery Akamai Tech přes port 80 (nezabezpečený protokol HTTP). Tato komunikace patří neúspěšným pokusům o stažení reklamy, která by se zobrazovala během instalace.

3.2.7 Shrnutí poznatků z analýzy instalace

Zkoumali jsme aktivitu procesu instalace programu Kappa. Sledovali jsme činnost na síti, na disku a v registrových klíčích. Instalační balíček se implicitně stahuje celý, následně se sám rozbalí a spustí se instalační program aplikace Kappa. Síťová komunikace při stahování probíhá přes protokol HTTP, což považujeme za závažný bezpečnostní nedostatek. Dále jsme zjistili, že instalátor i jeho importované knihovny používají ASLR a DEP, což je v pořádku. Zápis do registrových klíčů proběhl také korektně, ověřili jsme bezpečné použití COM objektu. Největším nalezeným bezpečnostním nedostatkem tedy je používání nezabezpečeného protokolu HTTP a neověřování svého digitálního podpisu při instalaci.

3.3 Analýza běhu programu

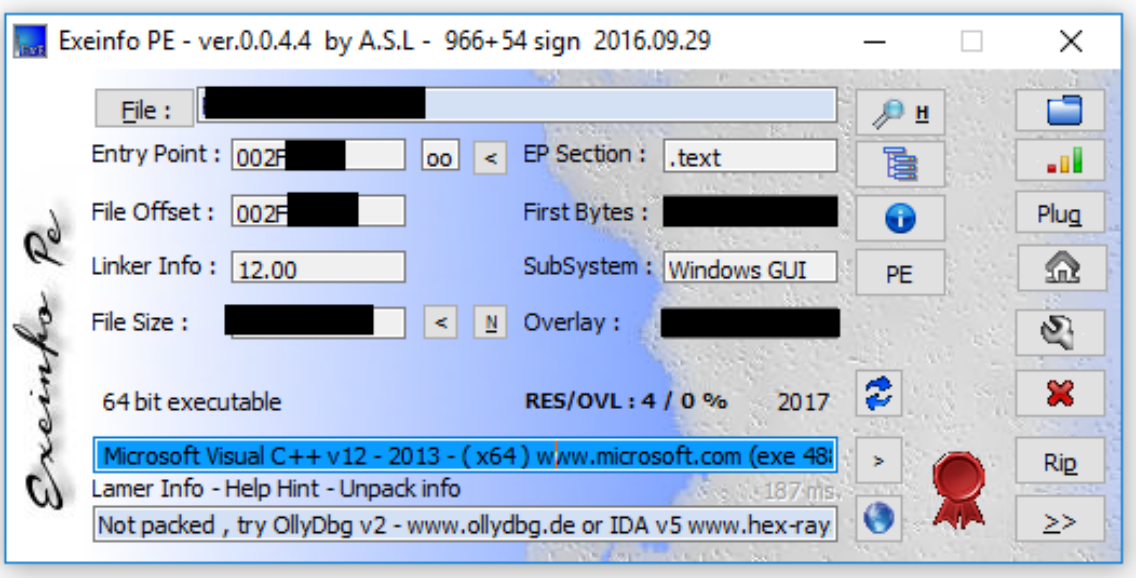
V této části budeme analyzovat chování programu Kappa při běžné činnosti, konkrétně budeme zkoumat prvky jeho grafického rozhraní, obvyklý běh a aktualizační proces. Nadále budeme využívat nástroje, které jsme popsali v předešlé kapitole. Běžné chování aplikace budeme monitorovat aplikací Process Monitor, na monitorování síťové aktivity využijeme program Wireshark a v případě potřeby zkoumat funkcionalitu nějaké části programu detailně budeme používat nástroje na reverzní inženýrství (2.2.2). Z manifestu zobrazeném v aplikaci Resource Hacker (2.2.1.4) zjišťujeme, že se program spouští s právy `asInvoker` [66].

3.3.1 Analýza běhu při startu počítače

Program Kappa se spouští automaticky se startem operačního systému, tudíž začneme s analýzou právě zde. K tomu využijeme nástroj Autoruns (2.1.2.3), který zkoumá registrové klíče související se startem operačního systému. Aplikace Autoruns po zadání jména programu do okénka filtrace vypíše veškeré registrové klíče, které souvisí se zkoumaným programem a se startem operačního systému. Ve výpisu můžeme pozorovat například knihovnu, kterou máme zaregistrovanou jako hodnotu klíče položky `InProcServer32` v rámci identifikace COM objektu v CLSID. Tyto klíče slouží jako identifikátory obslužných modulů (handlery) ke kontextovému menu. Ověřili jsme tedy, že operační systém při svém startu skutečně tento objekt spouští.

3.3.2 Analýza binárních souborů

Analyzovaný program má ve svém adresáři v Program Files mnoho binárních souborů, důkladné zkoumání všech nebude však naším cílem, a ani by to nebylo z časovým důvodů možné. Budeme se primárně věnovat binárnímu souboru, který ovládá grafické prostředí programu Kappa, a který se spouští při startu operačního systému. Při analýze programem ExeInfoPE (2.2.1.3) zjišťujeme, že byl program Kappa přeložený v Microsoft Visual Studiu 2013 překladačem MSVC++ verze 12. Analýzu tímto programem můžeme pozorovat na obrázku 3.6,



Obrázek 3.6: Zkoumání vlastností souboru programem ExeInfoPE

Dále můžeme analyzovat volaná API pomocí aplikace APImonitor (2.2.3.2), při monitorování spuštění programu se zobrazí několik tisíc záznamů o přístupech do knihoven, ale pokud nehledáme nějakou konkrétní informaci, tudíž analýza těchto záznamů příliš nepomůže. Tento nástroj tedy nevyužijeme a případně se k němu vrátíme později.

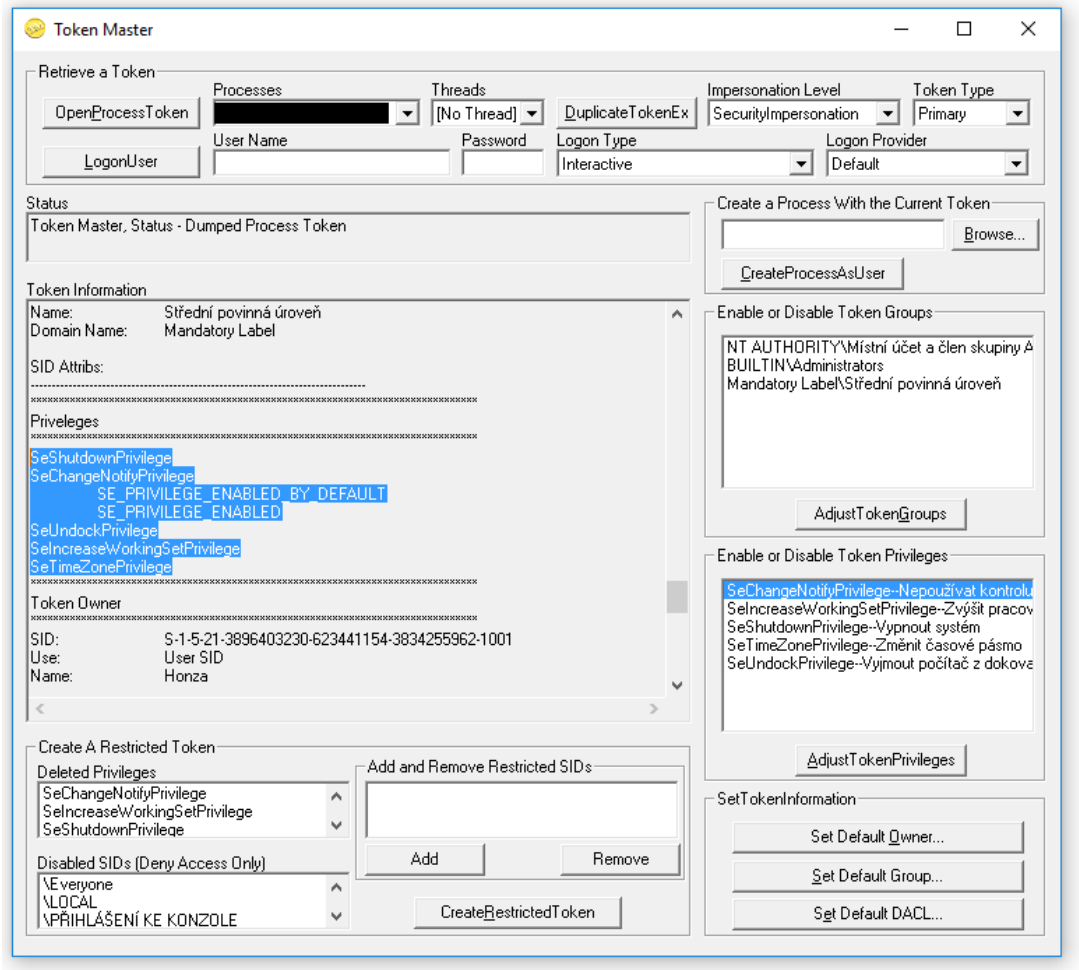
3.3.2.1 Analýza zabezpečení binárních souborů

Analýzu zabezpečení binárního spouštěcího souboru provádíme tak, že změníme byte, čímž dojde k zneplatnění digitálního podpisu [24]. Jelikož je možné změněný soubor bez problému spustit a program funguje i přesto standardně, můžeme usoudit, že program svůj digitální podpis neověřuje. Tedy digitální podpis jakožto bezpečnostní metoda nefunguje, protože program běží s běžnými právy uživatele a tedy neplatného podpisu si uživatel nemůže všimnout.

Pomocí aplikace CFF Explorer (2.2.1.1) prozkoumáme dynamicky linkované knihovny, při analýze budeme ověřovat, jestli spouštěcí binární soubor používá ASLR a DEP [50]. Binární soubor stejně jako všech 30 dynamicky linkovaných knihoven používá oba zmíněné ochranné mechanismy. Staticky linkované knihovny se pokusíme nalézt pomocí nástroje Strings (2.2.1.5). Nástroj vygeneruje tisíce řádků řetězců, žádný textový řetězec vedoucí k identifikaci staticky linkované knihovny se však nepodaří nalézt.

3.3.3 Analýza tokenu

Vlastnosti bezpečnostního tokenu procesu zkontrolujeme nástrojem TokenMaster (2.2.3.1). Na obrázku 3.7 jsou vypsaná práva procesu [36]. Zde vysvětlíme smysl těch, jejichž význam není zřejmý z názvu. Právo `SeChangeNotifyPrivilege` je implicitně povolené pro všechny uživatele a umožňuje tzv. Bypass traverse checking. To povoluje procesu přistoupit do složky, ke které mám povolení i přesto, že nemám práva přístupu k nadřazené složce. Tento případ obvykle nastává u přístupu do Program Files. Právo `SeUndockPrivilege` umožňuje procesu, aby odpojil počítač od dokovací stanice. Právo `SeIncreaseWorkingSet` povoluje procesu alokaci další paměti. Zjištěné používání práv je standardní.



Obrázek 3.7: Zkoumání bezpečnostního tokenu procesu programem Token Master

3.3.4 Analýza funkcionality grafického rozhraní

Grafické rozhraní programu Kappa umožňuje několik různých interakcí. Některé z nich zde nemůžeme popsat, abychom zachovali anonymitu aplikace. Vidíme zde panel s nápisem Sledovat Kappa a několik ikonek s odkazy na profily společnosti na různých sociálních sítích. Analýzou pomocí aplikace Wireshark (2.1.4.1) zjišťujeme, že tato komunikace probíhá po zabezpečeném kanálu. Tuto informaci ověříme pomocí aplikace Process Monitor, kde objevíme volaný příkaz `PID: 7492, Command line: "C:\Program Files (x86)\Mozilla Firefox\firefox.exe" -osint -url "https://www.facebook.com/Kappa"`.

Program dále umožňuje v nastavení změnit položku se zobrazováním reklam. Přestože je tato položka implicitně zaškrtnutá, žádné reklamy se nezobrazují. Při zkoumání logu z aplikace Process Monitor si všímáme, že v souvislosti s tím probíhá síťová komunikace procesu. Tuto komunikaci lokalizujeme v aplikaci Wireshark, kde nalezneme příčinu (viz obrázek 3.8). Program zjistí od DNS serveru IP adresu, chce stáhnout reklamy, ale přepojené servery Akamai mu vrací HTTP kód: `404: Soubor nenalezen` [25]. Dalším důkazem nestahování reklam je prázdná podsložka v adresáři `%USERPROFILE%\AppData\Local\Temp\Kappa`. Když by aplikace takto stahovala reklamy, museli bychom ověřit, že ošetřuje JavaScriptové vstupy.

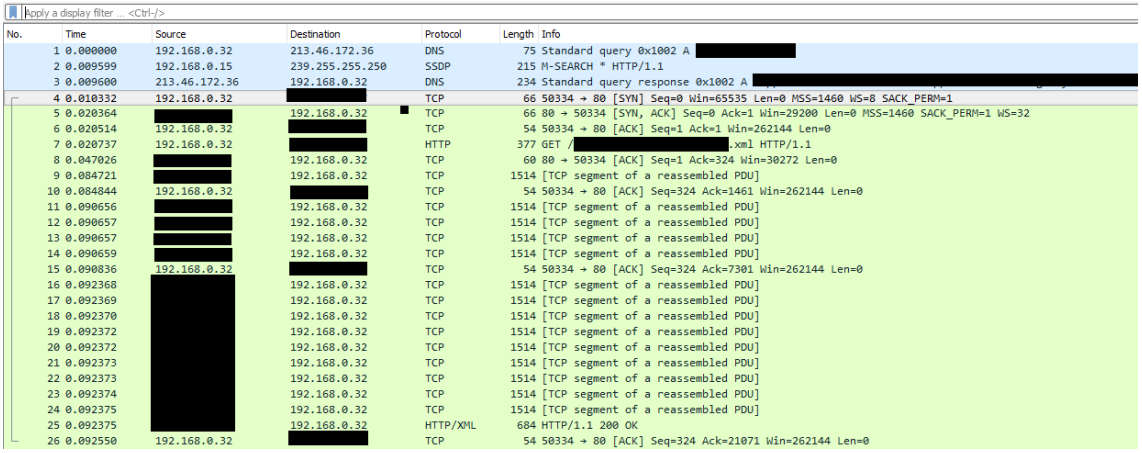
No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.168.0.32	213.46.172.36	DNS	76	Standard query 0xb07c A [redacted]
2	0.007723	213.46.172.36	192.168.0.32	DNS	163	Standard query response 0xb07c A [redacted] CNAME www2.[redacted] CNAME
3	0.014812	192.168.0.32	172.227.165.225	TCP	66	50184 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=8 SACK_PERM=1
4	0.032688	172.227.165.225	192.168.0.32	TCP	66	80 → 50184 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=32
5	0.032832	192.168.0.32	172.227.165.225	TCP	54	50184 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
6	0.032999	192.168.0.32	172.227.165.225	HTTP	370	GET /[redacted].json HTTP/1.1
7	0.048359	172.227.165.225	192.168.0.32	TCP	60	80 → 50184 [ACK] Seq=1 Ack=317 Win=30272 Len=0
[redacted]						
10	0.520212	172.227.165.225	192.168.0.32	TCP	222	[TCP segment of a reassembled PDU]
11	0.520309	192.168.0.32	172.227.165.225	TCP	54	50184 → 80 [ACK] Seq=317 Ack=169 Win=261976 Len=0
12	0.520764	192.168.0.32	172.227.165.225	TCP	54	50184 → 80 [FIN, ACK] Seq=317 Ack=169 Win=261976 Len=0
13	0.522408	172.227.165.225	192.168.0.32	HTTP	70	HTTP/1.1 404 Not Found (text/html)

Obrázek 3.8: Záznam síťové aktivity při stahování reklam z aplikace Wireshark

3.3.5 Analýza aktualizačního procesu

Aktualizační proces programu Kappa probíhá automaticky po spuštění operačního systému. Přibližně po pěti minutách můžeme pozorovat pomocí aplikací Wireshark a Process Monitor požadavek na servery společnosti Kappa. Nikde v nastavení jsme žádnou konkrétní hodnotu upravující prodlevu neobjevili, ani aplikace Autoruns nezobrazuje žádný registrový klíč, který by se s touto hodnotou mohl vázat. Při zkoumání aktualizačního XML konfiguračního souboru, který je uložený v Program Files\Kappa, jsme také žádný související parametr neobjevili.

Proces automatického vyhledávání aktualizací probíhá stejně jako proces při zmáčknutí tlačítka: Vyhledat aktualizace. Spustí se klasická síťová aktivita: DNS dotaz na překlad domény výrobce, HTTP GET požadavek na stažení aktualizačního XML souboru, stažení souboru a v programu Kappa jeho následné prozkoumání. V případě detekce nové verze se vyvolá upozornění o možnosti aktualizace. O průběhu se můžeme přesvědčit z následujícího obrázku 3.9.



Obrázek 3.9: Záznam síťové aktivity při vyhledání aktualizací z aplikace Wireshark

3.3.5.1 Analýza aktualizačního XML souboru

Soubor ve formátu XML se stáhne a uloží do složky `%USERPROFILE%\AppData\Local\Temp`. Tento soubor má poměrně specifickou strukturu, kvůli zachování anonymity programu Kappa zde nemůžeme popsat všechny jeho položky, ale aktualizační XML soubor má přibližně následující formu:

- Hlavička obsahuje informaci o verzi XML a kódování UTF-8.
- Nabídka podpůrného programu, který se stáhne v případě, že si aktualizační proces není jistý, jakou verzi programu použít.
- Odkazy na webové stránky používají nezabezpečený protokol HTTP [25], zatímco odkazy na stahované soubory používají zabezpečený protokol HTTPS [60].
- Verze produktů jsou rozdělené podle verze operačního systému, architektury a dalších parametrů počítače.
- Položky jednotlivých verzí mají mnoho parametrů, většina z nich se nepoužívá. Používá se přesný identifikátor verze programu, dále zkrácený identifikátor verze programu ve formátu `XX.Y.Z`. Každá položka obsahuje odkazy na instalátor nové verze programu v plné i minimální verzi.

Během analýzy aktualizačního XML souboru jsme si všimli, že soubor neobsahuje žádný obranný mechanismus např. formou haše pro ověření integrity stahovaného souboru. Tudíž aplikace Kappa přichází o nástroj na ověření stahovaného souboru.

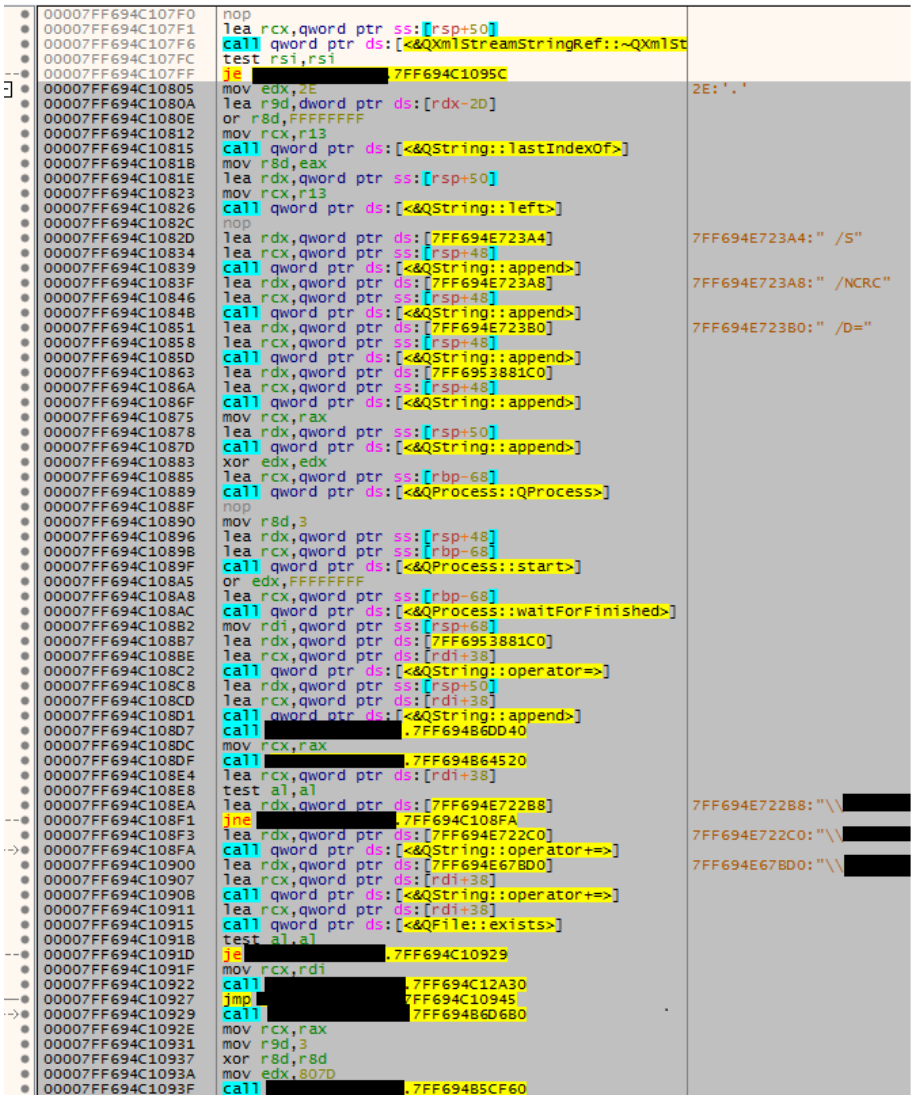
3.3.5.2 Průběh stahování aktualizačního instalátoru

Stahování aktualizačního samorozbalovacího balíku probíhá přes odkaz v aktualizačním XML souboru. Tento soubor uvádí odkaz na soubor pomocí HTTPS, ale v aplikaci Wireshark můžeme pozorovat požadavek stažení HTTP GET, tedy přes nezabezpečený kanál. Soubor se stahuje do podsložky `INetCache` ve složce `AppData\Local`, poté se informace přepisují do souboru uloženého ve složce pro stažené soubory `%USERPROFILE%\Downloads`. Stažený soubor je samorozbalovací archív, který se spouští v tichém režimu.

3.3.5.3 Analýza spuštění aktualizace

Parametry spuštění samorozbalovacího archívu nalezneme využitím technik reverzního inženýrství. Pomocí statické analýzy souboru (viz obrázek 3.10) jsme zjistili, že v této části program Kappa po stažení nového aktualizačního instalátoru spustí stažený soubor s následujícími parametry:

- /S – tento přepínač slouží pro spuštění samorozbalovacího archívu v tichém (Silent) režimu
- /NCRC – tento přepínač vypíná CRC kontrolu ověření archívu [41]
- /D=$cesta – tento přepínač nastavuje cestu, kam se má archív rozbalit. Program dále pomocí volání knihovní funkce `String_append` přidává dříve načtené hodnoty cesty. Archív se rozbalí do známého adresáře C:\Kappa.



Obrázek 3.10: Analýza spuštění instalátoru programem x64dbg

Program s těmito parametry spustí samorozbalovací archív, který se rozbalí do programem předepsaného adresáře. Tuto činnost můžeme pozorovat i na obrázku 3.10, konkrétně kolem volání knihovní funkce QProcess::Start. Poté program počká, až se archív rozbalí a spustí instalátor umístěný v několika podsložkách. Při zkoumání rozbalených souborů si můžeme všimnout, že v nejvyšší úrovni složky je program Setup.exe, který ovšem instalátor nepoužívá, protože používá raději instalátor umístěný hlouběji v adresářové struktuře.

Analýzou spouštěného instalátoru pomocí aplikace Resource Hacker (2.2.1.4) odhalíme v manifestu znovu požadavek `requireAdministrator`, tedy instalátor bude potřebovat zvýšená práva. V programu CFF Explorer (2.2.1.1) ověříme, že binární soubor spolu s importovanými knihovnami správně používá ASLR a DEP. Binární soubor je korektně digitálně podepsán vydavatelem, ověřený pomocí SHA1 a SHA256. Oba podpisy budeme považovat za bezpečné, ačkoli, jak už jsme ukázali dříve v této práci, otázka bezpečnosti SHA1 může být diskutabilní [55].

3.3.5.4 Přepínače instalátoru

Při spuštění instalace instalační program potřebuje zvýšit práva procesu. Okno UAC prozrazuje, že se instalátor spouští s parametry /BETA0 /[redacted]. První přepínač nastavuje parametry instalace:

- BETA 0 při volbě Expresní instalace
- BETA 1 při volbě Vlastní instalace
- Žádné další možnosti nastavení toho parametru jsme při analýze neobjevili

Zatímco využití prvního přepínače je nyní jasné, druhý přepínač je do jisté míry záhadou. Generování tohoto přepínače je v programu Kappa skryté, pro zjištění smyslu přepínače musíme staticky a dynamicky analyzovat spouštěný instalátor. Na obrázku 3.11 můžeme pozorovat volané instrukce při detekci nastavení přepínače v pozorovaném stavu při aktualizačním procesu. Analýzou tohoto kódu jsme zjistili, že program pomocí druhého přepínače upravuje svoje nastavení pro stažení dodatečných souborů.

Obrázek 3.11: Analýza druhého přepínače instalátoru programem x64dbg

3.3.5.5 Stažení JSON souboru

Po potvrzení zvýšení práv v UAC pozorujeme v aplikacích Wireshark a Process Monitor nejprve dva `HTTP GET` požadavky na soubor `error_codes.json`. Požadavky na webový server jsou odlišné, druhý požadavek je rozšířený o parametr: `Accept: */*`, ale jinak jsou požadavky identické a stahují identický obsah. Při zkoumání obsahu staženého `.json` souboru zjišťujeme, že se jedná o soubor, pomocí kterého program Kappa vysvětluje chybové hlášky v několika jazycích.

3.3.5.6 Průběh instalace aktualizace

Dále už pozorujeme obdobný průběh instalace, který jsme pozorovali předtím u čisté instalace (3.2). Není důvod zde duplikovat již zanalyzovaná a dříve popsaná zjištění.

3.3.6 Shrnutí poznatků z analýzy běhu programu

Pozorovali jsme běh programu Kappa. Nejdříve jsme pomocí programu Autoruns analyzovali registrové klíče, které souvisí se startem programu, protože se program Kappa spouští implicitně již při startu operačního systému. Dále jsme zkoumali nastavení, importované knihovny a zabezpečení automaticky spouštěného binárního souboru programu Kappa. V souvislosti s tím jsme se zabývali i analýzou bezpečnostního tokenu procesu, když jsme zkoumali s jakými právy běží proces. Poté jsme analyzovali funkcionalitu grafického rozhraní. Na závěr jsme zkoumali průběh aktualizačního procesu, kde jsme odhalili několik bezpečnostních pochybení.

Zjistili jsme, že většina komponent je implementována korektně podle bezpečnostních principů. Aplikace používá nízká práva pro svůj obvyklý běh, při potřebě zvýšení práv volá patřičné programy, které po skončení nutných akcí vypíná. Aplikace pracuje korektně i se soubory, nenašli jsme žádný soubor ve složce s nižšími přístupovými právy, který by mohl ovlivnit běh programu. Také nebyla zaznamenaná žádná úprava přístupových práv do složek.

Jediným slabým místem aplikace byl shledán proces aktualizace. Byla zjištěna absence jakéhokoli bezpečnostního prvku, který by ověřoval pravost serveru. Pravost stahovaného aktualizačního XML souboru není ověřována. Zjistili jsme, že tento soubor neobsahuje žádný bezpečnostní prvek na kontrolu stahovaných souborů a ani instalátor neověřuje digitální podpis svého certifikátu.

3.4 Vyhodnocení bezpečnostních nedostatků

V této kapitole jsme se doposud věnovali analýze instalace a běhu programu. Během analýzy instalace jsme příliš bezpečnostních pochybení nenašli. Jelikož možnosti útoku na instalaci jsou značně omezené oproti pravidelnému běhu (instalace probíhá pouze jednou), nebudeme se procesem čisté instalaci již dále zabývat. Nalezené nedostatky obvyklého běhu jsou ovšem závažné. Nyní představíme metodiku vyhodnocení, pomocí které tyto bezpečnostní nedostatky vyhodnotíme.

Pro vyhodnocování analýzy se obvykle používá Common Vulnerability Scoring System [40] alternativně pak Common Weakness Soring System [34]. My použijeme CVSS, který je rozšířenější a vyhodnocování pomocí této metriky je jednodušší. K samotnému výpočtu čísla využijeme kalkulačku dostupnou na stránkách vydavatele NIST [61]. Popíšeme položky, abychom při vyhodnocování snadněji rozuměli parametrům vektoru:

- Metrika proveditelnosti útoku
  - Vektory útoku (Attack Vector)
  - Složitost útoku (Attack Complexity)
  - Požadovaná práva (Privileges Required)
  - Interakce s uživatelem (User Interaction)
  - Rozsah (Scope)
- Metrika dopadu
  - Vliv na důvěrnost (Confidentiality)
  - Vliv na integritu (Integrity)
  - Vliv na dostupnost (Availability)

Jednotlivé položky mají různé možnosti ohodnocení. Ty nyní popíšeme vždy ve vzestupném směru podle vlivu na výsledné skóre. U vektorů útoku musíme vyhodnotit, zda zneužití nedostatku vyžaduje fyzický přístup (AV:P) k počítači, či zda postačí přístup do lokální sítě (AV:L), sousední sítě (AV:A) nebo je možné zaútočit na bezpečnostní nedostatek kdekoli přes síť (AV:N). Složitost útoku se rozlišuje pouze na vysokou (AC:H) a nízkou (AC:L). U požadovaných práv se vyhodnocuje, zda jsou k provedení potřeba práva administrátorská (PR:H), běžného uživatele (PR:L) či žádná (PR:N). Zneužití nedostatku může potřebovat k provedení nějakou interakci s uživatelem, v tom případě je ohodnocena položkou ve vektoru (UI:R) v opačném případě položkou (UI:N). Rozsah zneužití se dělí na nezměněný (S:U), kdy nedostatek ovlivní bezprostředně pouze zdroje přidružené k aplikaci, a na změněný (S:C), kdy chyba můžeme ovlivnit více zdrojů. Metriky dopadu zneužití nedostatku na důvěrnost (C), integritu (I) a dostupnost (A) mají stejné možnosti hodnocení: žádný (X:N), nízký (X:L) a vysoký (X:H). V případě nejasností při výběru jednotlivých položek můžeme nahlédnout do dokumentace vydavatele FIRST.org, Inc. [23]

Rozebírat podrobně jednotlivé vektory nalezených nedostatků nemá příliš smysl. Nalezneme je včetně vektorů a výsledného hodnocení v tabulce 3.1. Z tabulky je patrné, že některé nalezené nedostatky jsou opravdu závažné. Jedná se o posloupnost bezpečnostních nedostatků, které spolu souvisí. V rámci posouzení bezpečnosti aplikace Kappa proto navrhneme útok na aplikaci, jehož proveditelnost metodou Proof of Concept [59] dokážeme.

Útok je proveditelný napříč celou sítí (Internet), přesto jeho realizace bude podstatně snazší pouze v rámci lokální sítě. Při útoku na celou síť bychom museli být například poskytovatelem internetu (ISP) či koncový uzel sítě TOR [46]. Splnění této podmínky není samozřejmě triviální, ovšem o to větší účinek by poté mohl mít potenciální útok a zneužití bezpečnostních nedostatků.

Bezpečnostní nedostatek		
Vektor	**Hodnocení**	**Riziko**
Aplikace používá protokol HTTP		
`AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N`	6.5	Střední
Nekontroluje se pravost stahovaných souborů		
`AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:L`	9.6	Kritické
Nekontroluje se pravost aktualizačního serveru		
`AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:L`	9.6	Kritické
Instalátor aktualizace explicitně vypíná kontrolu CRC		
`AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:L/A:N`	4.3	Střední
Spouštěcí binární soubory neověřují svůj digitální podpis		
`AV:L/AC:L/PR:H/UI:R/S:C/C:H/I:H/A:L`	7.7	Vysoké

Tabulka 3.1: Tabulka nalezených bezpečnostních nedostatků a jejich rizik

3.5 Shrnutí

V této kapitole jsme zkoumali nejdříve průběh instalace a poté obvyklý běh aplikace Kappa. Z informací získaných během analýzy jsme sestrojili tabulku bezpečnostních nedostatků a rizik. Do tabulky jsme doplnili hodnocení z CVSS kalkulačky, která se běžně používá pro ohodnocování nedostatků. Některé nalezené nedostatky mají vysoké ohodnocení, tudíž je velké riziko, že je může útočník zneužít a získat kontrolu nad procesem. My se o to pokusíme v následujících kapitolách, kde nejdříve navrhneme modelový útok a dále se jej pokusíme realizovat, abychom ověřili jejich reálnost metodou Proof of Concept.

Kapitola 4

Návrh modelového útoku

V předešlé kapitole jsme se věnovali analýze aplikace Kappa. Cílem analýzy bylo posoudit bezpečnost aplikace. Zjistili jsme, že její tvůrci většinou ctili základní principy bezpečného návrhu systému a implementace [28], program pracuje korektně se vstupy (viz 2.1). Nalezli jsme však bezpečnostní nedostatky v procesu aktualizace softwaru. Objevili jsme hned několik vážných bezpečnostních pochybení v tomto procesu. Program při aktualizačním procesu komunikuje se serverem po nezabezpečeném kanálu HTTP [25], neověřuje pravost serveru, nekontroluje integritu stahovaných dat [5], vypíná kontrolu CRC [41] a neověřuje platnost digitálního podpisu [24].

Tato bezpečnostní pochybení budeme v této kapitole dále analyzovat a navrhneme teoretický útok na aplikaci Kappa a její aktualizační proces. Podobné útoky jsou poměrně běžné a nebezpečné, protože nevyžadují fyzický přístup k počítači. V našem případě to bude znamenat pouze požadavek na přístup do stejné lokální sítě. Z pozice poskytovatele internetu či koncového uzlu sítě TOR [46] by útok měl daleko větší účinnost.

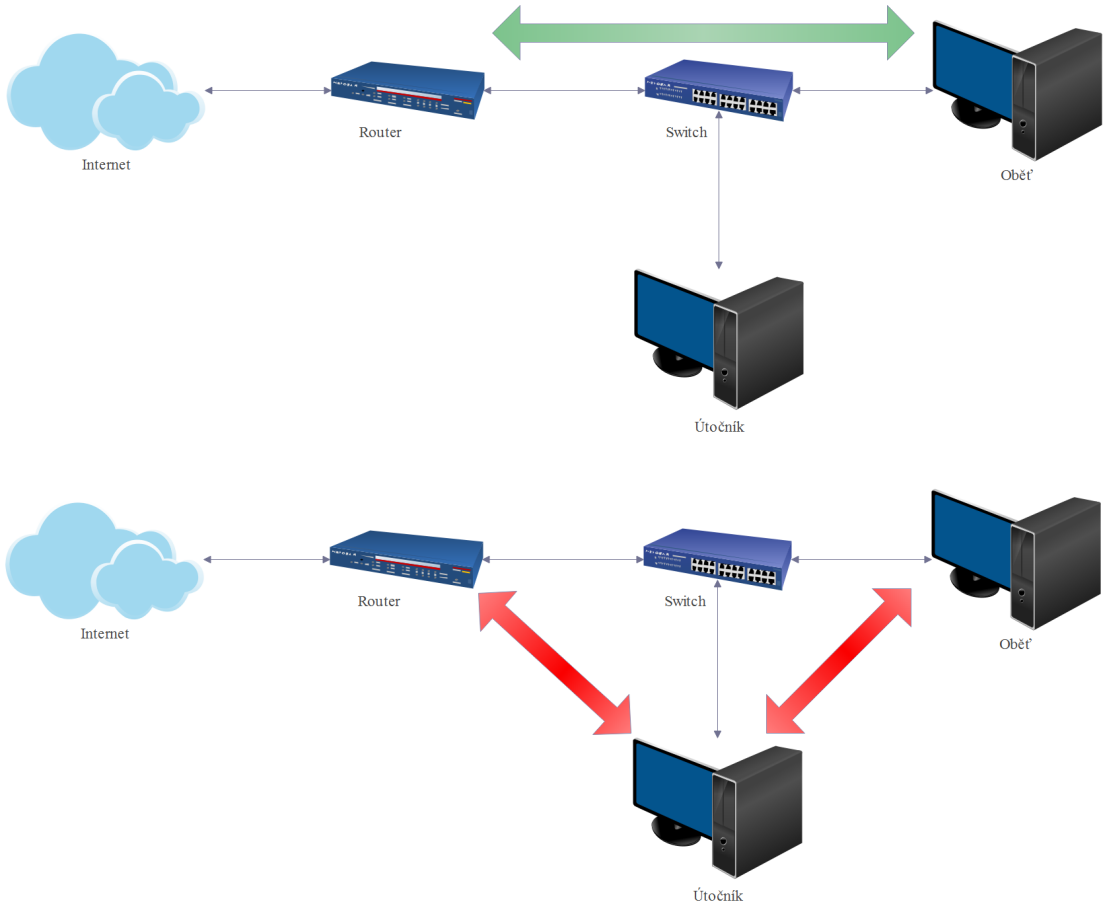
Síťové útoky jsou účinné, protože se pomocí nich dá útočit na všechny počítače v síti současně. Dále také proto, že síťová hierarchie obsahuje 7 úrovní podle OSI modelu [63] a je vysoká pravděpodobnost, že zabezpečení nějaké části bylo zanedbáno. Tyto metody útoků velmi dobře popisuje dokument od SANS [1], ve kterém můžeme zjistit přehledně a stručně, jaké metody útoků jsou běžné a použitelné na jednotlivých vrstvách OSI modelu. Další výhodou síťových útoků jsou špatné možnosti odhalení, pokud má uživatel přístup pouze k svému počítači.

V této kapitole představíme obecné metody útoku na nezabezpečenou komunikaci. Dále navrhneme, jak konkrétně by mohl vypadat útok na aplikaci pomocí metody Proof of Concept [59]. V následující kapitole realizujeme navržený útok a vyhodnotíme jeho dopad.

4.1 Útok na nezabezpečenou komunikaci

Aplikace Kappa komunikuje s aktualizačními servery po nezabezpečeném kanále. V této souvislosti jsme popsali využití HTTP protokolu, což je protokol nejvyšší vrstvy OSI modelu – aplikační. Nezabezpečení protokolů používajících tuto vrstvu má za následek, že je komunikace zranitelná útokem typu Man-in-the-Middle [29]. Útočník může získat v čitelné podobě data, která si aplikace vyměňuje se serverem.

Tento typ útoku je poměrně běžný a nebezpečný, protože se špatně detekuje. Útočník infiltruje komunikaci a snaží se ji nepozorovaně naslouchat. Útoku Man-in-the-Middle nezabrání použití HTTPS protokolu [60], pouze jej výrazně zkomplikuje, protože útočník bude muset podvrhnout certifikát serveru. Schéma pro tento útok je zobrazen na obrázku 4.1.



Obrázek 4.1: Schéma útoku Man-in-the-Middle

Při nezabezpečené komunikaci stačí útočníkovi přesvědčit oběť, aby mu posílala své pakety. Útočník se rozhodne, jak s nimi dále naloží, a buď je dále standardní cestou přepošle nebo zahodí. K tomu může využít útočník například tzv. ARP otrávení [56] nebo útok na bezdrátové spojení pomocí deautentizačních rámců [39]. K Man-in-the-Middle útokům na šifrovanou komunikaci se používá např. HTTPS únos spojení [3]. Složitost tohoto útoku je vyšší a úspěšnost závisí na správné implementaci HTTP Strict Transport Security (HSTS) [37].

4.1.1 Útok na protokol ARP

Adress Resolution Protocol používáme pro překlad IP adres na MAC adresy [44]. V rámci OSI modelu tedy tento protokol pracuje mezi druhou a třetí vrstvou. Protokol se využívá v IPv4, u IPv6 se používá obdobně koncipovaný protokol NDP [38]. Protokol ARP použijeme například ve chvíli, kdy potřebujeme zjistit, kam máme v rámci sítě poslat IP datagram. Funkcionalita je jednoduchá, odesílatel vyšle ARP požadavek pomocí broadcastu po celé síti na všechny MAC adresy. Všichni příjemci si zapíší do své ARP cache informace o odesílateli a na tento požadavek odpoví pouze hledaný adresát, vlastník IP adresy. Ten pošle ve své ARP odpovědi obě adresy druhé a třetí vrstvy, tedy MAC a IP adresu.

4.1.1.1 Otrávení paměti protokolu ARP

Útok spočívá v otrávení záznamů protokolu ARP obětí v síti útočníkem. Ten k tomu využije tzv. ARP podvrhnutí [32], kdy útočník v rámci sítě odešle podvržené ARP požadavky. Pomocí těchto požadavků přesvědčí všechna zařízení v síti, aby komunikovala s ním místo výchozí brány. Z toho důvodu musí mít útočník přístup do lokální sítě přes infikovaný počítač nebo fyzicky. Poté útočník rozešle podvržené ARP dotazy, čeká a odposlouchává komunikaci, která nyní proudí přes něj. Tuto situaci můžeme pozorovat na obrázku 4.1 se schématem Man-in-the-Middle, protože otrávení cache ARP je zde pouze prostředek, jak dosáhnout tohoto útoku na obrázku. Ve chvíli, kdy útočník zachytí nějaký datagram, může:

- datagram zahodit a odpovědět odesílateli sám;
- datagram nechat projít dál a změnit až odpověď serveru.

Obě možnosti mají samozřejmě stejný výsledek, tedy že oběť v dobré víře komunikuje s domnělým serverem, ale ve skutečnosti dostává podvržené odpovědi. Důvod pro použití první možnosti může být ten, že útočník nechce, aby původní adresát vůbec tušil, že se ho někdo snažil kontaktovat. V tom případě tedy útočník nepřeposílá zprávu a sám se pokusí odpovědět oběti.

Lepší využití druhé možnosti může být například ve chvíli, kdy útočník neví přesně, co odesílatel žádá od adresáta a jakou strukturu by měla mít odpověď. Obvykle mezi sebou komunikují programy, které mají pevně danou strukturu očekávané odpovědi, aby ji mohly správně přeložit a rozpoznat. Útočník může použít technik reverzního inženýrství, aby tuto strukturu odhalil. Ve většině případů však bude jednodušší využít zde zmíněnou druhou možnost, případně dříve odposlechnout a zaznamenat tento proces a zaútočit až při jeho opakování.

V případě naší práce by mohl útok pomocí otrávení cache protokolu ARP probíhat tak, že po otrávení bychom čekali na zaslaný HTTP GET na servery programu Kappa s požadavkem na aktualizační XML soubor. Pomocí tohoto soubor aplikace Kappa zjistí, zda nevyšla nová verze. Soubor bychom tedy mohli podvrhnout a donutit program, aby místo aktualizačního programu spustil připravený škodlivý program. K tomu bychom mohli využít nástroje oblíbené v Kali Linux, dostupné i na ostatních linuxových distribucích a Microsoft Windows: Scapy [7] či Ettercap [52].

4.1.1.2 Ochrana proti útoku na protokol ARP

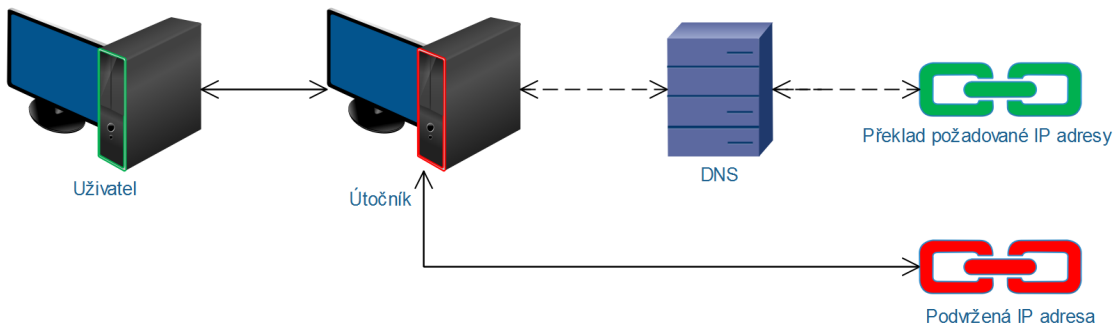
V souvislosti s útokem na protokol ARP zde představíme i několik způsobů, jak útoku zabránit. Nejjednodušší je nastavit statické používání ARP protokolu. Vstupy se nastaví na mód pouze čtení, a poté se musí namapovat na každém počítači jeho vlastní ARP tabulka, kterou není možné měnit. Použití tohoto zabezpečení je možné v malé síti s minimální fluktuací zařízení připojených do sítě. Ve velkých sítích je použití statického protokolu ARP nereálné.

Další možností je použití nástrojů na detekci ARP podvrhnutí na síťových zařízeních. Naprostá většina moderních kvalitních přepínačů podobnými nástroji disponuje. Tyto nástroje kontrolují ARP komunikaci a křížově prověřují odpovědi na požadavky. Případně kontrolují vysílané ARP dotazy, a podezřele aktivním prvkům sítě přepínače vypnou přístup (port shutdown). Softwarové nástroje se obvykle instalují na DHCP servery či ethernetové přepínače. Možností obrany koncových zařízení je instalace programů typu ARP AntiSpoofer[19] nebo úprava registrových klíčů, pomocí které docílíme omezení možností uprav tabulky paměti protokolu ARP.

4.1.2 Útok na protokol DNS

V předchozí části jsme popsali, jak se používá otrava cache protokolu ARP v případě Man-in-the-Middle útoku. Využití ARP cache otrávení se dá využít u útoku na cache protokolu DNS. Domanin Name System (zkráceně DNS) [57] je hierarchický systém doménových jmen. Tento systém má za úkol překládat doménová jména na IP adresy.

Hierarchické zapojení sítě slouží k tomu, aby se záznamy nemusely obnovovat v reálném čase, ale současně bylo možné v nich co nejrychleji hledat a překládat doménová jména všech zaregistrovaných položek. Z toho důvodu jsou servery zapojeny do pyramidové hierarchie. V případě, že DNS server neumí přeložit doménové jméno, přepošle dotaz na server o úroveň výše. Takto dotaz může doputovat až ke kořenovému DNS serveru, který zná všechna jména. Pro naše potřeby nám postačí vědět, že existuje taková služba a protokol na aplikační vrstvě, který využívá na transportní vrstvě protokol UDP [58].



Obrázek 4.2: Schéma útoku na DNS

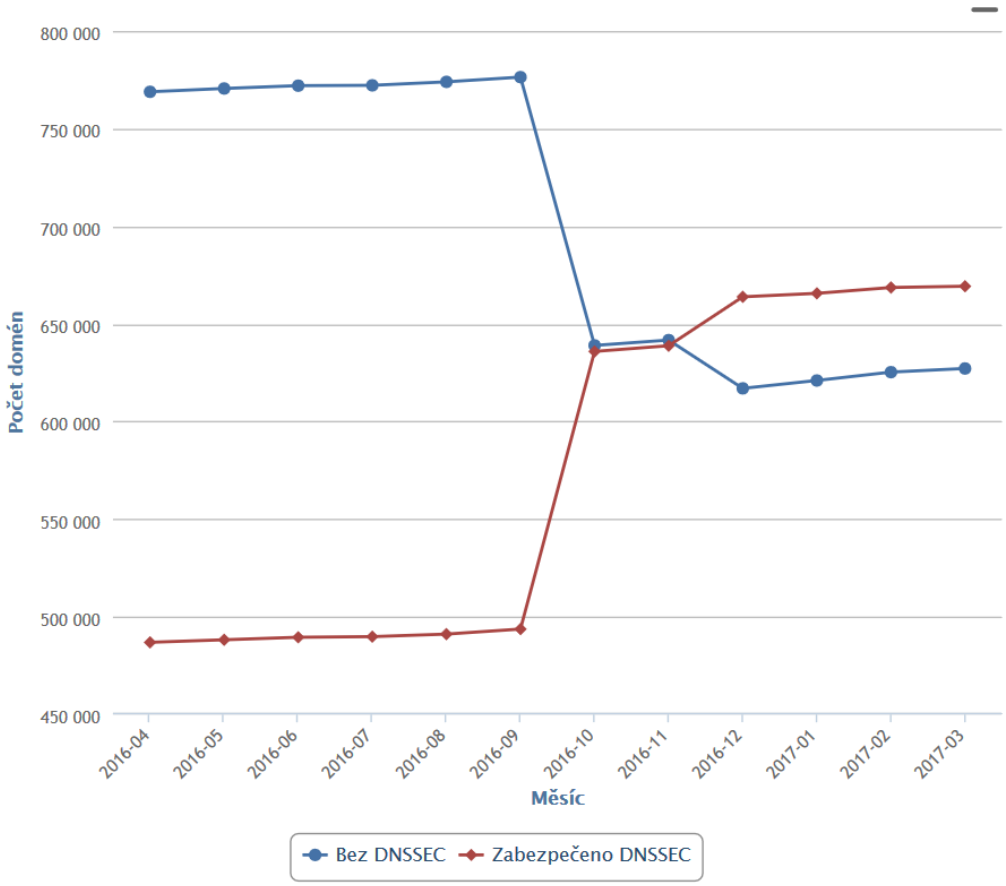
[19] Odkaz na webové stránky aplikace AntiSpoofer: `https://sourceforge.net/projects/arpantispoofer`

4.1.2.1 Otrávení DNS cache

V roli útočníka budeme chtít pomocí útoku na protokol DNS přesvědčit oběť, že na IP adrese našeho serveru se nachází server s jeho hledaným doménovým jménem. Přesvědčovacích technik máme několik, nejjednodušší bude využít již výše popsaná ARP podvrhnutí, zachytit požadavek na DNS server a vrátit mu IP adresu našeho serveru. V tu chvíli se k němu oběť připojí s důvěrou, že jedná se správným serverem. Tomuto útoku se říká otrávení DNS cache.

4.1.2.2 Ochrana proti útoku na protokol DNS

Zabezpečení proti otravě DNS cache je relativně jednoduché, stačí využít zabezpečenou verzi téhož protokolu — DNSSEC [33]. Tento protokol byl vytvořen speciálně proto, aby zabránil těmto typům útoku. DNSSEC funguje na podobném principu jako původní DNS, pouze navíc jednotlivé zprávy šifruje pomocí asymetrické kryptografie. Pro šifrování a dešifrování obsahu se tedy používají různé klíče [14]. Bohužel celosvětové statistiky statdns.com[20] ukazují, že se tento protokol příliš nezačal používat. V tomto případě je Česká republika výjimkou a v používání DNSSEC protokolu je jedna z nejlepších na světě. Na obrázku 4.3 můžeme pozorovat vývoj používání zabezpečené verze v ČR.



Obrázek 4.3: Vývoj používání protokolu DNSSEC, zdroj: nic.cz

[20] Odkaz na webové stránky statdns: `https://www.statdns.com`

4.1.3 Shrnutí

V této sekci jsme představili různé metody útoku na počítače v rámci sítě, což v našem případě bude vyžadovat fyzický přístup k síti. Popsali jsme teoreticky metodu Man-in-the-Middle a možnosti realizace pomocí otravy protokolů ARP a DNS. U těchto metod jsme ve zkratce popsali jejich funkcionalitu, účel a možnosti obrany proti zneužití.

4.2 Návrh útoku na aktualizační proces

V předešlé kapitole jsme provedli rozsáhlou analýzu instalace a běhu programu Kappa. Při zkoumání samotné instalace jsme objevili nedostatky zabezpečení v používání nezabezpečeného protokolu HTTP a neověřování digitálního podpisu instalačního souboru. Při zkoumání běhu programu jsme objevili posloupnost bezpečnostních chyb v nezabezpečení aktualizačního procesu. Zjistili jsme, že program neověřuje pravost aktualizačního serveru, komunikace není šifrovaná, program neověřuje pravost stahovaných dat a dokonce explicitně vypíná kontrolu CRC [41]. Na začátku této kapitoly jsme popsali dvě možné realizace útoků na nezabezpečenou síť pomocí útoku Man-in-the-Middle. Nyní již pouze zbývá popsat, jak tyto útoky použít při útoku na aplikaci Kappa.

Návrh útoku je v teoretické rovině jednoduchý. Útok na protokol DNS byl vyhodnocen jako jednodušší, tudíž jej použijeme. Při analýze jsme zachytili veškerou komunikaci a poměrně přesně analyzovali chování aplikace. Nyní bude stačit pouze vybavit server požadovanými soubory, aby vyhověl všem požadavkům, které na něj aplikace bude mít. Pokud jsme nepřehlédli v analýze nějaký bezpečnostní prvek, tak by se pomocí podvrženého aktualizačního XML souboru mělo podařit program nechat stáhnout nový instalátor, který se následně spustí s administrátorskými právy.

4.2.1 Úprava aktualizačního XML souboru

Při analýze aktualizačního XML souboru (3.3.5.1) jsme pečlivě zkoumali strukturu stahovaného souboru. Zjistili jsme, že zde chybí jakékoli ověření stahovaných souborů pomocí haše. Nepoužívání bezpečnostních prvků samozřejmě zlehčuje práci při vyrábění vlastního aktualizačního souboru. Při jeho úpravě musíme změnit identifikační čísla verze instalátoru, aby byl program Kappa na počítači oběti přesvědčen, že je dostupná nová verze, a zobrazilo se upozornění o aktualizacích.

Identifikační čísla jsou v aktualizačním XML souboru dvě, první s názvem „internal“ označuje vnitřní přesnou verzi programu. Číslo program Kappa porovná se svým aktuálním číslem a v případě vyššího čísla v aktualizačním souboru se aktivuje možnost aktualizace. Druhé číslo se zobrazuje uživateli a nemá vliv na chod programu. Když však chceme uživatele přimět, aby souhlasil s aktualizacemi, je vhodné i toto číslo navýšit, aby i on měl pocit, že instaluje novější verzi. Dále zde musíme umístit odkaz na instalátor. Jeho výrobu popíšeme v následující sekci této kapitoly.

4.2.2 Příprava instalátoru

V předchozí kapitole jsme popsali, jak vypadá instalátor (3.3.5.3) a s jakými parametry se spouští (3.3.5.4). Instalátor je binární spustitelný soubor typu SFX, který se sám rozbalí do přepínačem definované složky. Naším cílem bude tedy vytvořit v následující kapitole takový samorozbalovací archív, který bude schopný se rozbalit do složky a bude obsahovat v podsložce spouštěcí instalační program.

Instalátor má funkční digitální podpis [24] a jeho modifikací bychom jej zneplatnili, čehož by si mohl všimnout uživatel při schválení elevace práv v okně UAC. Z toho důvodu bychom při reálném útoku museli nahradit nějakou importovanou knihovnu námi nově vytvořenou knihovnou. Veškerou funkcionalitu původní knihovny bychom však museli zachovat, proto bychom museli v nové knihovně implementovat veškeré exportované symboly původní knihovny pomocí přesměrování na původní knihovnu. Tímto bychom zachovali plnou funkcionalitu a současně bychom mohli spustit jakýkoli kód skrz novou knihovnu s administrátorskými právy. Instalátor tedy bude znovu zabalený archív, který jsme dříve legitimní cestou stáhli a rozbalili.

4.3 Shrnutí

V této kapitole jsme aplikovali poznatky z analýzy a v teoretické rovině jsme navrhli útok na aplikaci Kappa formou Proof of Concept. Definovali jsme parametry útoku typu Man-in-the-Middle a jeho dosažení pomocí otravy ARP cache. Dále jsme popsali metodu otrávení cache protokolu DNS, díky které se počítač útočníka bude vydávat za server komunikující s počítačem oběti. V kapitole jsme se věnovali také konkrétním přípravám útoku na aktualizační proces, kde jsme popsali plánovanou úpravu aktualizačního XML souboru a přípravu vlastního instalátoru.

Kapitola 5

Realizace

V předchozí kapitole jsme popsali modelový útok na aplikaci Kappa. Tento útok je cílený na dříve nalezená bezpečnostní pochybení v aktualizačním procesu. V této kapitole modelový útok realizujeme. Nejdříve se pokusíme získat kontrolu nad nezabezpečenou komunikací programu Kappa mezi počítačem oběti a servery. O to se pokusíme pomocí Man-in-the-Middle útoku, který realizujeme pomocí podvržení protokolu ARP (viz. 4.1.1) a otravy cache protokolu DNS (viz. 4.1.2). Schéma a teoretický základ pro tento útok jsme popsali v předešlé kapitole. Útok realizujeme pomocí programu Ettercap [52], který byl na podobnou činnost přímo zkonstruován. Tímto přimějeme oběť, aby komunikovala s naším serverem.

Dalším krokem bude příprava aktualizačního XML souboru a instalátoru. I tyto části byly v předešlých kapitolách důkladně analyzovány a navrženy. Aktualizační soubor upravíme, aby obsahoval novější verzi a umístíme do něj odkaz na pozměněný instalátor. Ten vytvoříme pomocí nástroje NSIS [47], který je využíván i tvůrci aplikace. Jako potvrzení Proof of Concept bude považována situace, kdy se podaří spustit vlastní instalátor s administrátorskými právy. Tímto by mělo být prokázáno riziko bezpečnostního nedostatku aplikace.

5.1 Příprava testovacího prostředí

Realizace útoku je testována na dvou osobních počítačích s OS Windows 10. Útočník dostal od DHCP serveru přidělenou IP adresu 192.168.0.30, oběť dostala adresu 192.168.0.32. Útočník bude mít navíc k dispozici Kali Linux [19] nainstalovaný ve Virtual Boxu [11]. Oba počítače budou připojeny do stejné sítě pomocí běžně používaného směrovače TECHNICOLOR TC7200. Vlastní DNS server nebude instalován, protože bude stejně hned na začátku odříznut od komunikace. Budeme využívat DNS server poskytovatele internetu, společnosti UPC Česká republika, s.r.o. Pomocí příkazu `nslookup` v příkazové řádce zjistíme, že se služba `svchost` připojuje k DNS serverům poskytovatele cz-prg01a-dns01.chello.cz na adrese 213.46.172.36 a cz-prg01a-dns02.chello.cz na adrese 213.46.172.37. Počítače i směrovač jsou ve výchozím nastavení, nebyla měněna žádná konfigurace ani instalována žádná zařízení typu IDS, která by komplikovala realizaci Man-in-the-Middle. Celý průběh útoku samozřejmě budeme pečlivě zkoumat a analyzovat.

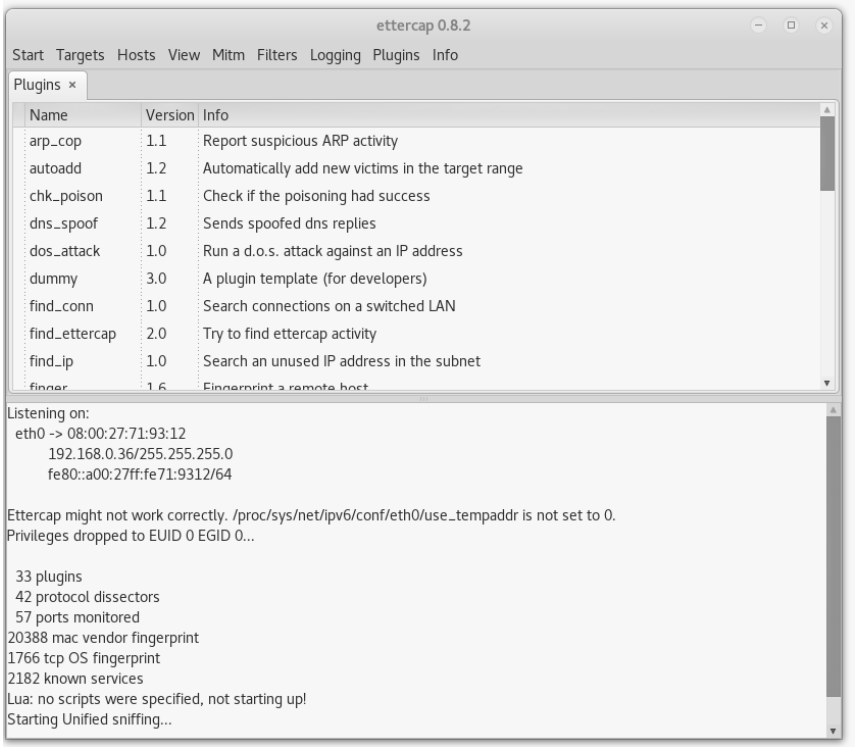
5.2 Příprava útoku na nezabezpečenou komunikaci

Během analýzy jsme zjistili, že aplikace nepoužívá šifrované spojení, ale komunikuje pomocí protokolu HTTP [25]. Věříme, že se podařilo odchytit a zanalyzovat přesnou formu a strukturu komunikace aplikace s aktualizačním serverem, který v této kapitole podvrhneme. K realizaci Man-in-the-Middle útoku použijeme podvrhnutí odpovědi protokolu ARP, pomocí kterého zachytíme DNS dotaz na překlad jména serveru, a podvrhneme odpověď.

Celá situace nastane ve chvíli, kdy aplikace Kappa po přibližně pěti minutách od startu operačního systému kontroluje nové aktualizace. Aplikace pošle pomocí služby svchost DNS dotaz s doménovým jménem serverů společnosti Kappa. Útočník tento požadavek zachytí a odpoví, že on sám je ten server. Oběť začne s podvrženým serverem komunikovat tradičním způsobem, po navázání komunikace pomocí trojcestného podání rukou následuje požadavek HTTP GET kvůli stažení aktualizačního XML souboru. Útok bude realizován pomocí nástroje Ettercap [52], a tak jej zde nyní představíme.

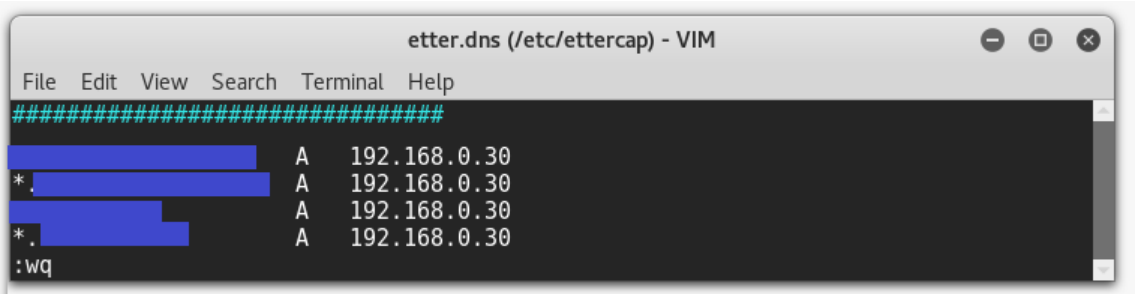
5.2.1 Použití aplikace Ettercap

Ettercap je aplikace s otevřeným zdrojovým kódem dostupná pod licencí GNU. Byla vytvořena přímo pro realizaci Man-in-the-Middle útok na lokální síti. Nástroj je určený pro linuxové distribuce a můžeme jej ovládat z příkazové řádky, pomoci GUI či přes ncurses [18]. Aplikace umožňuje odposlouchávání, filtrování a analýzu mnoha protokolů. Toho dosáhne nastavením síťového rozhraní do tzv. promiskuitního režimu, který umožňuje naslouchat komunikaci, která není určena pro toto konkrétní rozhraní.



Obrázek 5.1: Grafické rozhraní nástroje Ettercap

Z obrázku 5.1 můžeme vidět, že nabízených 33 zásuvných modulů dává zajímavé spektrum možností, navíc tento nástroj umožňuje realizovat útok Man-in-the-Middle pomocí otrávení cache ARP, ICMP přesměrování, krádeže portu, DHCP podvrhnutí či otrávení cache NDP. V našem případě použijeme již výše zmíněnou kombinaci otrávení cache protkolů DNS a ARP. Ačkoli jsme si ukázali pro názornost grafické rozhraní aplikace Ettercap, pro samotný útok použijeme verzi ovládanou přes příkazový řádek, protože se budou snáze zaznamenávat jednotlivé kroky. Pomocí programu VIM otevřeme soubor `/etc/ettercap/etter.dns`, který upravuje nastavení filtrů Ettercapu pro DNS útok, a přidáme řádky týkající se serverů programu Kappa. Konkrétní úpravy můžeme vidět na obrázku 5.2. S druhým serverem komunikuje aplikace až na základě odkazu na soubor v aktualizačním XML souboru, není tedy nutné ho tam přidávat.

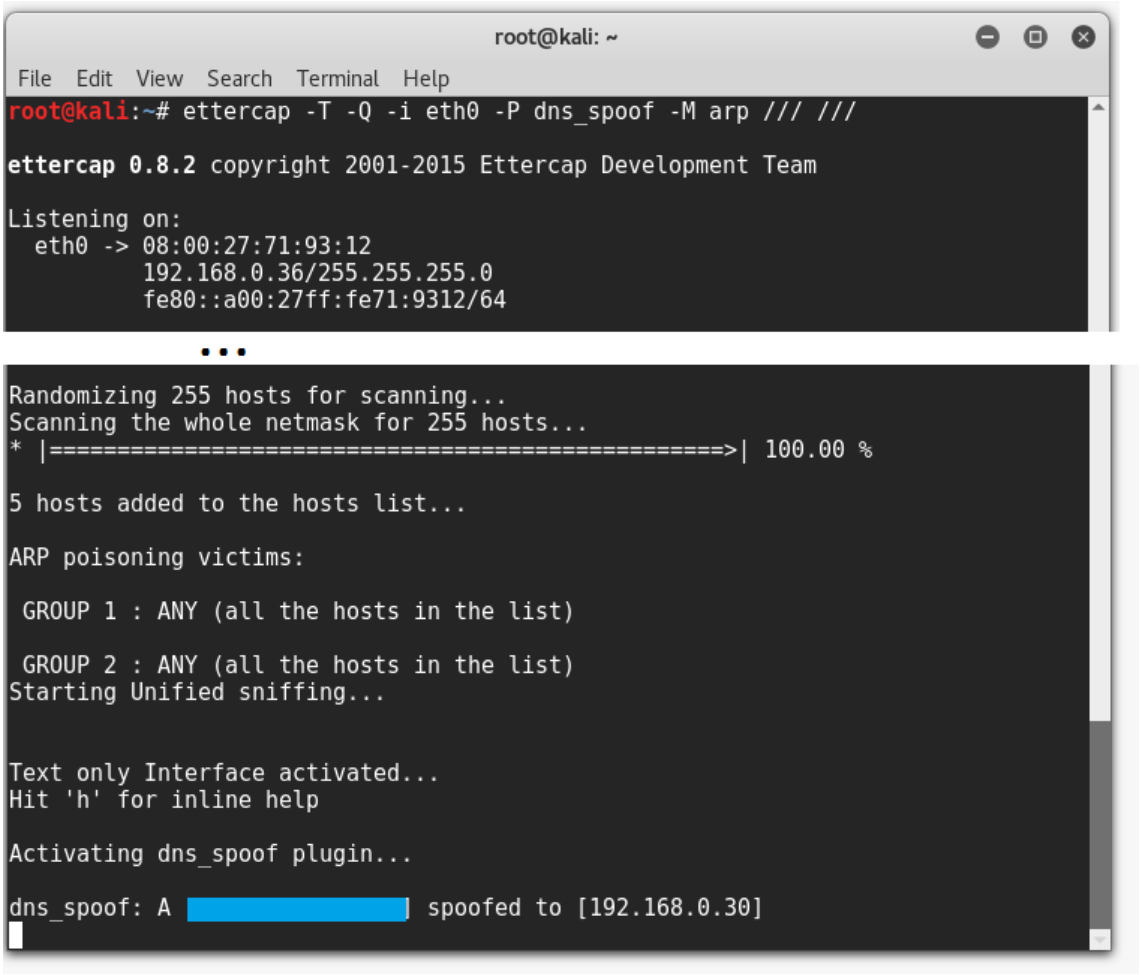


Obrázek 5.2: Úprava filtru DNS v programu Ettercap

Nyní je již vše připraveno a můžeme spustit Ettercap z příkazové řádky pomocí příkazu: `ettercap -T -Q -i eth0 -P dns_spoof -M arp /// ///`, jehož spuštění můžeme sledovat na obrázku 5.3. Přepínače [13] mají následující úlohu:

- T nastavuje textové rozhraní;
- Q nastavuje mód superquiet, Ettercap nevypisuje uživatele ani hesla, tyto informace pouze ukládá;
- i specifikuje síťové rozhraní, které budeme používat, toto rozhraní po dobu útoku přepne do promiskuitního režimu, který umožňuje odposlouchávat cizí komunikaci;
- P umožňuje použít zásuvný modul, v našem případě dns_spoof;
- M vybírá metodu Man-in-the-Middle útoku, v našem případě útok na ARP;
- Dvě trojice lomítek umožňují specifikovat zdrojovou či cílovou IP adresu. V našem případě tuto možnost necháme nevyplněnou, protože chceme naslouchat odchozí i příchozí komunikaci počítače oběti (celé sítě).

Na obrázku 5.3 pozorujeme úspěšnou realizaci DNS podvržení. V aplikaci Wireshark na počítači oběti si všímáme DNS dotazu a podvržené odpovědi. Poté aplikace naváže komunikaci na transportní vrstvě (tzv. trojcestné podání rukou [58]) paketem TCP [SYN]. Odpovědi v podobě TCP [SYN, ACK] se mu však nedostane, protože jsme zatím na počítači útočníka nespustili HTTP server, ani nenaprogramovali soket naslouchající na portu 80. Pro jednoduchost nebudeme používat soket, ale nainstalujeme WAMP server [51].

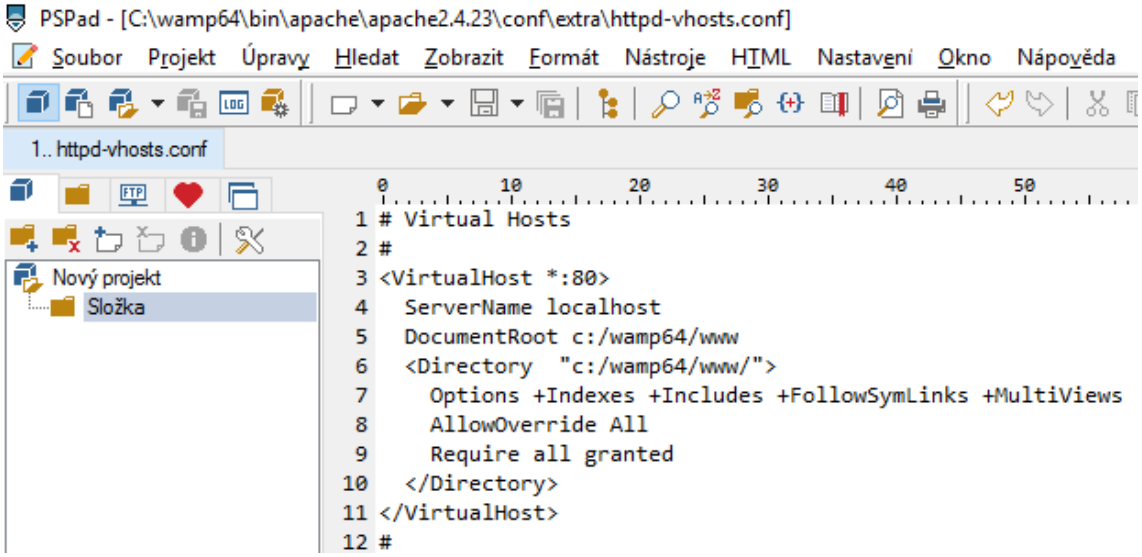


Obrázek 5.3: Spuštění útoku pomocí Ettercap z příkazové řádky

5.2.2 Instalace a nastavení HTTP serveru

V tuto chvíli potřebujeme nějaký HTTP server na počítači útočníka, jako nejjednodušší varianta se jeví použití WAMP serveru, který je volně dostupný pod licencí GNU GPL. Server se skládá z Apache webového serveru, OpenSSL, MySQL a PHP. Díky tomu budeme mít k ruce nástroj, se kterým bychom si poradili i v případě, když bychom narazili např. na šifrovanou komunikaci. Instalační program stáhneme ze stránek WAMP serveru. Program instalujeme s implicitním nastavením, výsledek se rozbalí do složky C:\wamp64. V této složce najdeme mnoho souborů a složek. Pro nás bude důležitá složka www, která slouží jako ROOT adresář Apache serveru.

Před otestováním našeho serveru ještě musíme změnit nastavení přístupu do složky www, jinak bude server vracet na GET požadavky odpověď `403: Přístup odepřen`. V rámci našeho testovacího prostředí si můžeme dovolit nastavit přístupy v souboru httpd-vhosts.conf na hodnotu `Require all granted`. Přesné nastavení i cestu k souboru můžeme vidět na obrázku 5.4.



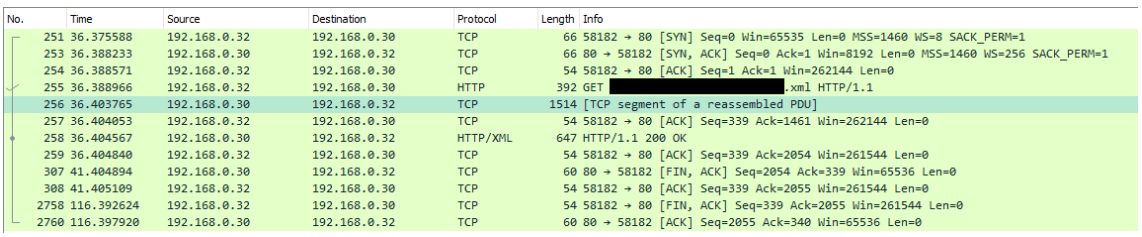
Obrázek 5.4: Nastavení WAMP serveru

5.2.3 Příprava souborů pro server

Ze závěrů analýzy víme, že HTTP server bude muset umět reagovat na HTTP požadavky na aktualizační soubory ve formátu XML a souboru s chybovými hláškami ve formátu json. Tyto soubory umístíme do patřičné podsložky v adresáři www. Přesná cesta k souborům musí sedět s cestami v HTTP požadavcích aplikace. Zatímco json soubor můžeme beze změny zkopírovat, aktualizační XML soubor otevřeme, upravíme hodnoty tykající se interní a veřejné verze aplikace. Dále zde můžeme přepsat doménové jméno serveru B na doménové jméno serveru A. Ze serveru B se implicitně stahuje instalátor a přepsáním nebudeme muset pomocí nástroje Ettercap padělat DNS dotaz i pro něj.

5.2.4 Testování správného nastavení HTTP serveru

Nyní by již mělo být vše připraveno. Spuštění serveru bude vyžadovat administrátorské oprávnění, protože program bude používat port 80. V záznamech z aplikace Wireshark na obrázku 5.5 můžeme vidět důkaz, že se podařilo podvrhnout server a přimět aplikaci Kappa, aby z něj stáhla aktualizační XML soubor. Nyní již je potřeba pouze vytvořit funkční instalátor.

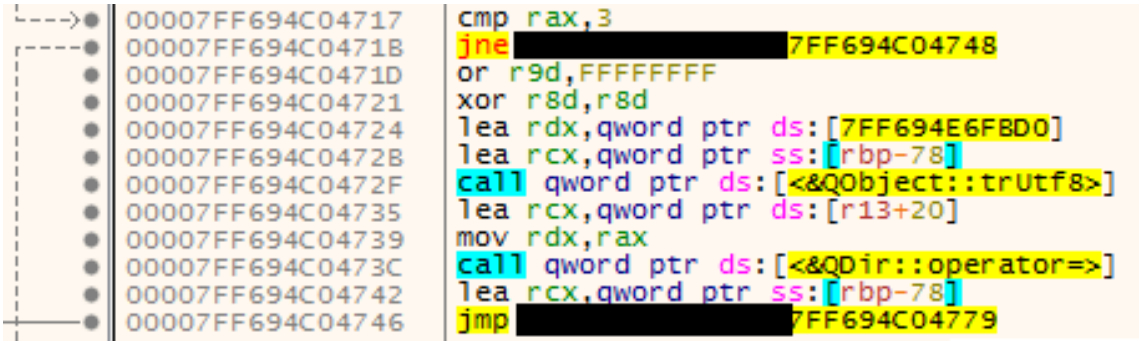


Obrázek 5.5: Záznam z aplikace Wireshark — úspěšné nahrání podvrženého aktualizačního XML souboru

5.2.5 Výroba instalátoru

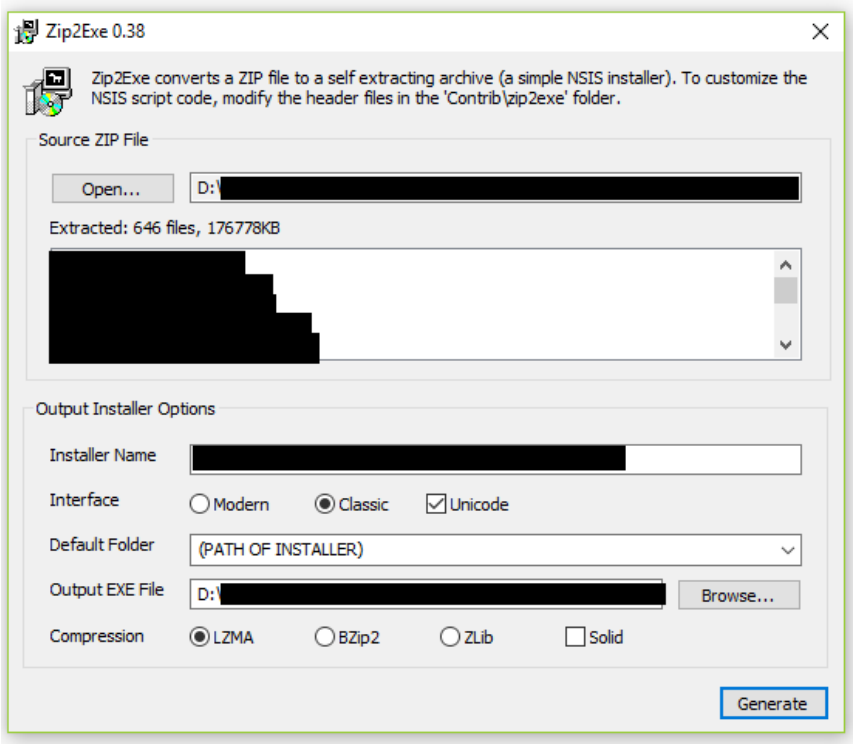
Během analýzy automatického aktualizačního procesu (3.3.5.3) jsme sledovali proces instalace a manipulace se samorozbalovacím archívem SFX. Zjistili jsme, že archív byl vytvořen pomocí aplikace NSIS [47]. Samorozbalovací archív se spouští dle manifestu s právy `asInvoker`. Rozbalení probíhá v tichém režimu kvůli přepínači `/S`, nekontroluje se CRC [41] kvůli použití přepínače `/NCRC` a instalátor se rozbalí do stanovené složky díky přepínači `/D=$cesta`. Po rozbalení se spustí v podsložce instalátor. Ten je podepsaný digitálním podpisem a dle manifestu bude při spuštění vyžadovat elevaci práv pomocí UAC.

Aplikaci bychom mohli předložit místo samorozbalovacího archívu nějaký jiný program. V tom případě by se při zavolání knihovní funkce `QProcess::start` [12] (obrázek 3.10) program spustil. Jelikož by však později aplikace nemohla najít v podadresáři instalátor, nastavila by příznaky registrů r9 a r8 (posledních 5 instrukcí na stejném obrázku 3.10). Později by aplikace tyto příznaky objevila a vypsala chybovou hlášku. Toto chování můžeme pozorovat na následujícím obrázku 5.6.



Obrázek 5.6: Detekce chyby a vypsání chybové hlášky v programu x64dbg

Pro demonstraci Proof of Concept postačí, když se podaří spustit námi zabalený archív, který se rozbalí do předepsané složky a bude obsahovat v podsložce patřičný instalátor. Rozbalíme standardní cestou originální stažený instalátor. V tuto chvíli bychom při reálném útoku mohli například nahradit nějakou knihovnu naší vlastní knihovnou, ale to není v případě Proof of Concept potřeba. Rozbalenou složku zabalíme pomocí libovolného archivačního nástroje a uložíme výsledek ve formátu .zip. V menu NSIS vybereme v sekci Compiler (překladač) možnost Installer based on ZIP file, rozhraní NSIS skriptu můžeme vidět na obrázku 5.7.



Obrázek 5.7: Výroba samorozbalovacího archívu SFX ze ZIP archivu v aplikaci NSIS

Takto vytvoříme staronový samorozbalovací archív. Skript je však implicitně nakonfigurován tak, že vyžaduje zvýšená práva už pro rozbalení, což je rozdíl oproti původnímu archívu. Kvůli tomu program Kappa vyhodí chybu, protože běží s běžnými právy uživatele a využité knihovní funkce `Qt` jsou použity tak, že zvýšení práv v tuto chvíli nepovolují. Abychom tuto informaci ověřili, musíme pomocí reverzního inženýrství zkoumat knihovnu `Qt5Core`, konkrétně volané funkce:

- konstruktor `QProcess::QProcess(QObject *parent = Q_NULLPTR)`
- funkci `QProcess::start(const QString &program, const QStringList &arguments, OpenMode mode = ReadWrite)`; pomocí této funkce se spouští nový proces, využitím technik reverzního inženýrství a nastudováním dokumentace jsme analyzovali volané parametry
  1. parametr je adresa knihovny QT5CORE.dll v registru rcx
  2. parametr je adresa na zásobník, která odkazuje na cestu k rozbalenému instalátoru, uložena v registru rdx
  3. parametr je hodnota 3, znamenající otevírací mód `ReadWrite` v registru r8
- metodu `bool QProcess::waitForFinished(int msecs = 30000)`

Z analýzy pomocí reverzního inženýrství vyplývá, že se nikde ve volaných funkcích nevyskytuje volání `ShellExecute`, který be umožňovalo zvýšení práv. Obdobně se za tímto účelem při práci s knihovnou `Qt` využívá metoda `startDetached` místo zde použité funkce `start`.

Nejjednodušším řešením je použít nástroj Resource Hacker, vyexportovat manifest z původní složky a nahrát ho do našeho nového podvrženého archívu. Jakmile tak učiníme, archív se spustí s právy `asInvoker` a program Kappa již bez problému spustí tento podvržený program. Poté se spustí instalace aktualizace s administrátorskými právy, během které by mohlo dojít ke spuštění škodlivého kódu skrz modifikovanou knihovnu.

5.3 Realizace útoku

V tuto chvíli již máme vše připraveno pro útok na aplikaci. Pro přehlednost zde ještě připomeneme seznam kroků potřebných při přípravě útoku, analýzu již zde zmiňovat nebudeme.

1. Vytvoření filtrů pro útok na DNS
2. Úprava aktualizačního XML souboru
3. Příprava HTTP serveru
4. Výroba instalátoru

Všechny tyto kroky jsme popsali podrobně již dříve v této kapitole. Nyní již stačí pouze spustit WAMP (HTTP server), pomocí Ettercap spustit útok na protokol DNS pomocí ARP podvrhování, restartovat počítač oběti a počkat přibližně pět minut na dotaz o automatických aktualizacích či vyvolat tuto činnost manuálně v grafickém rozhraní programu. Následně se podaří nahrát pozměněný instalátor do počítače oběti, rozbalí se a spustí instalátor s administrátorskými právy.

5.4 Vyhodnocení útoku

Popsaný postup se podařilo provést, když jsme na počítači oběti jsme spustili upravený instalátor. Můžeme potvrdit, že nalezené nedostatky v zabezpečení aplikace Kappa představují reálné bezpečnostní riziko.

5.5 Návrh bezpečnostních protiopatření

V této kapitole jsme ověřili na reálných zařízeních, že povaha nalezených nedostatků v zabezpečení umožňuje jejich zneužití. Metodou Proof of Concept jsme prokázali správnost navrženého modelu útoku. Nyní je nutné navrhnout bezpečností protiopatření na nalezené bezpečnostní nedostatky, které povedou ke snížení či úplnému zamezení možností útoku na aplikaci Kappa. Možnostem zabezpečení protokolů ARP (4.1.1) a DNS (4.1.2) jsme se věnovali v rámci jejich popisu v předešlé kapitole. Nyní je tedy ještě potřeba doporučit způsoby zabezpečení samotné aplikace Kappa. Jelikož se jedná o posloupnost bezpečnostních pochybení, projdeme proces aktualizace postupně od začátku a navrhneme řešení na jednotlivé bezpečnostní nedostatky.

1. Používání DNSSEC; pro překlad doménového jména se používá protokol DNS, již dříve jsme popsali výhody použití šifrované komunikace, proto bychom zde doporučili použití šifrovaného DNSSEC. Jeho nasazení není jednoduché, protože pro správnou funkčnost je nutné, aby podnikly patřičné kroky všechny strany [30], konkrétně:

    - Koncový uživatel
    - Poskytovatel internetových služeb
    - Poskytovatel DNS serverů

2. Používání zabezpečeného spojení; aplikace Kappa komunikuje po nezabezpečeném kanále HTTP na aplikační vrstvě, díky tomu má odposlouchávající útočník naprostý přehled o komunikaci mezi uživateli a servery. Může zde odposlechnout požadavek na stažení aktualizačního XML souboru či samotných aktualizačních programů. Šifrováním budou možnosti odposlechu ztíženy.

3. Přidání haše do aktualizačního XML soubor; soubor neobsahuje haš stahovaného souboru, tudíž aplikace nemůže ověřit, zda stáhla a spustila soubor, o kterém se domnívá, že ho stáhla z originálního serveru.

4. Implementace ověření digitálního podpisu instalačního programu; aplikace je správně digitálně podepsaná, ale chybí zde ověření podpisu za běhu programu. Správně implementovaný instalátor by měl kontrolovat svou vlastní haš.

5. Zapnutí kontroly instalátoru pomocí CRC. Zjistili jsme při rozbalování použití přepínače `/NCRC`, který explicitně vypíná CRC ověření, samorozbalovací archív v tu chvíli nemůže ověřit, zda nějaká část stahovaných dat nebyla pozměněna.

6. Kontrola certifikátu webového serveru s aktualizacemi

Tento výčet návrhu bezpečnostních opatření není konečný, obsahuje však základní řešení největších nalezených bezpečnostních pochybení, která vedou k funkčnímu útoku na aplikaci.

5.6 Shrnutí

V této kapitole jsme navázali na kapitolu předchozí a v ní popsaný teoretický model útoku jsme zde realizovali. Stanovili jsme jako cíl metody Proof of Concept spuštění námi zabaleného instalačního balíčku s administrátorskými právy. Při reálném útoku bychom ještě museli nahradit a přemapovat nějakou knihovnu. Originální instalátor není vhodné přepisovat, protože by pozměněný instalátor nebyl podepsaný společností programu Kappa. Ke spuštění instalátoru jsme museli vyrobit samorozbalovací instalační balíček, který ve správné podsložce po rozbalení bude mít patřičný program, pomocí kterého aplikace Kappa spouští instalaci. Dále jsme museli připravit filtr pro Ettercap, upravit vhodně aktualizační XML soubor a nainstalovat HTTP server. Funkcionalitu jednotlivých částí jsme testovali průběžně.

Realizace útoku se ukázala jako možná, skutečně se podařilo relativně jednoduše zneužít bezpečnostní nedostatky aplikace a spustit na počítači oběti námi upravený archív a následně instalátor s administrátorskými právy. Demonstrovali jsme, že ohodnocená rizika nalezených bezpečnostních nedostatků mají po právu vysoké hodnoty. Jejich zneužití je relativně jednoduché a dopad může mít fatální.

Kapitola 6

Závěr

Tato práce se zabývá bezpečnostní studií aplikace. Cílem bylo seznámení se s nástroji pro analýzu softwarového programu za účelem posouzení jeho bezpečnosti. Při monitorování činnosti počítače v aplikaci Wireshark byl objeven proces tohoto programu, který komunikuje se servery pomocí nezabezpečeného protokolu HTTP. Skutečné jméno aplikace zde nemůžeme uvést, protože se jedná o dílo veřejné a je zde popsán skutečný útok na aplikaci včetně detailního postupu v jednotlivých krocích. Z toho důvodu byly veškeré názvy a informace vedoucí k identifikaci programu přejmenovány na symbol Kappa, nebo byly začerněny již ve zdrojovém kódu vždy stejným blokem o délce pěti znaků.

Práce se dělí celkem na šest kapitol a začíná motivačním úvodem. V analytické části se práce zabývá rozdělením nástrojů podle jejich zaměření. Nástroje jsme rozdělili podle použití pro analýzu vstupů programu a na nástroje pro pochopení funkcionality programu. Skupina nástrojů zkoumající vstupy slouží k analýze práce s registrovými klíči, soubory a síťovou komunikací. Při představování nástrojů na pochopení funkcionality programů jsme popsali různé nástroje na ladění, získávání informací z binárních souborů či vlastností bezpečnostních tokenů. Celkem jsme v této kapitole popsali přes dvacet nástrojů pro analýzu, ze kterých jsme si poté vybrali přibližně patnáct nástrojů, které jsme používali během analýzy.

V rámci analýzy aplikace jsme pomocí vybraných nástrojů sledovali nejdříve průběh instalace, a poté obvyklý běh aplikace Kappa. U obou částí jsme pečlivě zkoumali vstupy programů, konkrétně při analýze Windows Registry jsme analyzovali registrové klíče, které ovlivňují program při startu operačního systému, mají potenciálně slabé ACL či manipulují s COM objekty. Při analýze manipulace soubory jsme zjišťovali, zda se žádný důležitý binární či konfigurační soubor neukládá do složek s nižšími přístupovými právy. Dále jsme kontrolovali, zda má program Kappa nastavená korektně práva ve svém bezpečnostním tokenu, a zda nedošlo během sledované činnosti ke změnám ACL v přístupech ke složkám. Při analýze síťové komunikace jsme objevili použití nezabezpečeného protokolu HTTP, dále pak sérii bezpečnostních pochybení v aktualizačním procesu. V rámci analýzy binárních souborů jsme ověřovali správné použití ASLR a DEP u samotných binárních souborů stejně jako u importovaných knihoven. Pomocí technik reverzního inženýrství jsme hledali v programu použití bezpečnostních prvků, obvykle jsme žádné použití nenalezli, což jsme na běhu aplikace ověřovali. Na konci kapitoly byly vyhodnoceny nalezené nedostatky.

Ve čtvrté kapitole jsme se zabývali návrhem možného útoku. Jelikož jsme hlavní bezpečnostní nedostatky objevili v síťové komunikaci v rámci aktualizačního procesu, věnovali jsme se primárně útoku na ni. Popsali jsme teoretické principy útoku Man-in-the-Middle.

Dále jsme popsali jeho realizaci pomocí útoku na protokoly ARP a DNS. Navrhli jsme úpravu aktualizačního XML souboru a popsali vytvoření samorozbalovacího instalátoru pomocí nástroje NSIS.

V páté kapitole jsme realizovali útok podle navrženého modelu ze čtvrté kapitoly. Nainstalovali jsme WAMP server, upravili aktualizační XML soubor aktualizace, připravili filtry pro Man-in-the-Middle útok pomocí Ettercap, vytvořili nový samorozbalovací instalační archív a následně provedli útok.

Realizace proběhla úspěšně a tudíž můžeme konstatovat, že se podařilo realizovat Proof of Concept zneužití bezpečnostních nedostatků. Také jsme v rámci této kapitoly definovali možné bezpečnostní protiopatření proti nalezeným nedostatkům. Diskutovali jsme možnosti zabezpečení aplikace Kappa, dále možnosti ochrany proti útoku na protokoly ARP a DNS. Z hlediska bezpečnosti doporučujeme nepoužívat aktualizační proces, novou verzi vždy stahovat přímo z internetových stránek a pokaždé provést čistou instalaci aplikace.

Bibliografie

[1] Advanced Technologies, Inc. Escal Institute of. *Applying the OSI Seven Layer Network Model To Information Security*. [online]. [cit. 20. dubna 2017]. 2004. URL: https://www.sans.org/reading-room/whitepapers/protocols/applying-osi-layer-network-model-information-security-1309.

[2] Batra, R. *API Monitor*. [online]. [cit. 20. dubna 2017]. 2012. URL: http://www.rohitab.com/apimonitor.

[3] Blažek, R. B. *Intrusion Techniques: Man-in-the-Middle, WiFi Deauthentication*. [online]. [cit. 20. dubna 2017]. 2016. URL: https://edux.fit.cvut.cz/courses/MI-SIB.16/_media/lectures/mi-sib-lec02-intrusiontechniques.pdf.

[4] Borge, S. *Automating Windows Administration*. 1. vyd. Apress, 2004.

[5] Boritz, J. E. *IS Practitioners' Views on Core Concepts of Information Integrity*. 1. vyd. Information Systems Audit a Control Association, 2003.

[6] Box, D. *Essential COM*. 1. vyd. Pearson P T R, 1998.

[7] Burns, B. et al. *Security Power Tools*. 1. vyd. O'Reilly Media, 2007.

[8] Chappe, L. a Combs, G. *Wireshark Network Analysis*. 2. vyd. Podbooks.com, Llc., 2010.

[9] Chikofsky, E. J. a Cross, J. H. *Reverse engineering and design recovery: A taxonomy*. [online]. [cit. 20. dubna 2017]. 1990. URL: http://win.ua.ac.be/~lore/Research/Chikofsky1990-Taxonomy.pdf.

[10] Coldewey, D. *This security camera was infected by malware 98 seconds after it was plugged in*. [online], TechCrunch. [cit. 20. dubna 2017]. 2016. URL: http://social.techcrunch.com/2016/11/18/this-security-camera-was-infected-by-malware-in-98-seconds-after-it-was-plugged-in.

[11] Colvin, H. *VirtualBox: An Ultimate Guide Book on Virtualization with VirtualBox*. 1. vyd. CreateSpace Independent Publishing Platform, 2015.

[12] Company, The Qt. *Library Qt Documentation*. [online]. [cit. 20. dubna 2017]. 2017. URL: http://doc.qt.io/.

[13] Crenshaw, A. *Manual Reference Pages - ETTERCAP (8)*. [online]. [cit. 20. dubna 2017]. 2016. URL: http://www.irongeek.com/i.php?page=backtrack-3-man/ettercap.

[14] CZ.NIC, z. s. p. o. *Jak funguje DNSSEC*. [online]. [cit. 20. dubna 2017]. 2017. URL: https://www.dnssec.cz/page/444/jak-funguje-dnssec/.

[15] CZ.NIC, z. s. p. o. *Rekapitulace (D)DOS útoků ze dnů 4. 3. – 7. 3.* [online]. [cit. 20. dubna 2017]. 2013. URL: https://www.csirt.cz/files/csirt/Rekapitulace-utoky-20120311.pdf.

[16] Dang, B. et al. *Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation*. 1. vyd. Wiley, 2014.

[17] Danseglio, M. *Securing Windows Server 2003*. 1. vyd. O'Reilly Media, Inc., 2004.

[18] Dickey, T. E. *NCURSES - New Curses*. [online]. [cit. 20. dubna 2017]. 2014. URL: http://invisible-island.net/ncurses/.

[19] Dieterle, D. W. *Basic Security Testing with Kali Linux*. 1. vyd. CreateSpace Independent Publishing Platform, 2014.

[20] Eagle, Ch. *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*. 2. vyd. No Starch Press, 2011.

[21] Eilam, E. *Reversing: Secrets of Reverse Engineering*. 1. vyd. Wiley, 2005.

[22] ENOM, INC. *Mapping Mirai: A Botnet Case Study*. [online]. [cit. 20. dubna 2017]. 2016. URL: https://www.malwaretech.com/2016/10/mapping-mirai-a-botnet-case-study.html.

[23] FIRST.org, Inc. *CVSS v3.0 User Guide (v1.4)*. [online]. [cit. 20. dubna 2017]. 2017. URL: https://www.first.org/cvss/user-guide.

[24] Goldreich, O. *Foundations of Cryptography: Volume 1, Basic Tools*. 1. vyd. Cambridge University Press, 2001.

[25] Gourley, D. et al. *HTTP: The Definitive Guide*. 2. vyd. O'Reilly Media, 2002.

[26] Graff, M. G. a Wyk, K. R. van. *Secure Coding: Principles and Practices*. 1. vyd. O'Reilly Media, 2003.

[27] Hornbeck, J.C. *Process Monitor – Hands-On Labs and Examples*. [online]. [cit. 20. dubna 2017]. 2008. URL: https://blogs.technet.microsoft.com/appv/2008/01/24/process-monitor-hands-on-labs-and-examples/.

[28] Howard, M. a LeBlanc, D. *Writing Secure Code for Windows Vista*. 1. vyd. Microsoft Press, 2010.

[29] Kaufman, Ch. et al. *Network Security: Private Communications in a Public World (Radia Perlman Series in Computer Networking and Security)*. 1. vyd. Prentice Hall, 2002.

[30] Kolkman, O. *DNSSEC HOWTO, a tutorial in disguise*. [online]. [cit. 20. dubna 2017]. 2009. URL: https://www.nlnetlabs.nl/publications/dnssec_howto/.

[31] Krebs, B. *Hacked Cameras, DVRs Powered Today's Massive Internet Outage*. [online]. [cit. 20. dubna 2017]. 2016. URL: https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/.

[32] Lockhart, A. *Network Security Hacks: Tips & Tools for Protecting Your Privacy*. 1. vyd. O'Reilly Media, 2006.

[33] Lucas, M. W. *DNSSEC Mastery: Securing the Domain Name System with BIND*. 1. vyd. CreateSpace Independent Publishing Platform, 2013.

[34] Martin, B. *MITRE: Common Weakness Scoring System*. [online]. [cit. 20. dubna 2017]. 2014. URL: https://cwe.mitre.org/cwss/cwss_v1.0.1.html.

[35] Matsui, M. *Advances in Cryptology*. 1. vyd. AsiaCrypt, 2009.

[36] Microsoft Corp. *Privilege Constants*. [online]. [cit. 20. dubna 2017]. 2017. URL: https://msdn.microsoft.com/en-us/library/windows/desktop/bb530716(v=vs.85).aspx.

[37] Mutton, P. *95% of HTTPS servers vulnerable to trivial MITM attacks*. [online]. [cit. 20. dubna 2017]. 2016. URL: https://news.netcraft.com/archives/2016/03/17/95-of-https-servers-vulnerable-to-trivial-mitm-attacks.html.

[38] Narten, T. et al. *RFC 4861: Neighbor Discovery for IP version 6 (IPv6)*. [online]. [cit. 20. dubna 2017]. 2007. URL: https://tools.ietf.org/html/rfc4861.

[39] Nguyen, T. D. et al. *A Lightweight Solution for Defending Against Deauthentication/ Disassociation Attacks on 802.11 Networks*. [online]. [cit. 20. dubna 2017]. 2008. URL: http://ieeexplore.ieee.org/document/4674211/.

[40] Ou, X. a Singhal, A. *Quantitative Security Risk Assessment of Enterprise Networks*. 1. vyd. springer, 2011.

[41] Peterson, W. W. a Brown, D. T. *Cyclic Codes for Error Detection*. 1. vyd. IEEE, 1961.

[42] Pietrek, M. *Peering Inside the PE: A Tour of the Win32 Portable Executable File Format*. [online]. [cit. 20. dubna 2017]. 1994. URL: https://msdn.microsoft.com/en-us/library/ms809762.aspx.

[43] Pistelli, D. *CFF Explorer*. [online]. [cit. 20. dubna 2017]. 2012. URL: http://www.ntcore.com/exsuite.php.

[44] Plummer, D. C. *RFC: 826, An Ethernet Address Resolution Protocol*. [online]. [cit. 20. dubna 2017]. 1982. URL: https://tools.ietf.org/html/rfc826.

[45] Richter, J. a Clark, J. *Programming Server-Side Applications for Microsoft Windows 2000 (Microsoft Programming)*. 1. vyd. Microsoft Press, 2000.

[46] Rogers, B. *Tor: Beginners to Expert Guide to Accessing the Dark Net, TOR Browsing, and Remaining Anonymous Online*. 1. vyd. CreateSpace Independent Publishing Platform, 2017.

[47] Russell, J. a Cohn, R. *Nullsoft Scriptable Install System*. 1. vyd. Book on Demand, 2012.

[48] Russinovich, M. a Margosis, A. *Troubleshooting with the Windows Sysinternals Tools*. 2. vyd. Microsoft Press, 2016.

[49] Ryan. *Yet Another Infosec Blog: Packed Executables*. [online]. [cit. 20. dubna 2017]. 2006. URL: http://yaisb.blogspot.cz/2006/07/packed-executables.html.

[50] S. Huntley. *Windows Exploitation Course: Stack and Heap Overflows*. 1. vyd. CreateSpace Independent Publishing Platform, 2015.

[51] Safari Books Online, LLC. *Learning WAMP Server Development*. [DVD]. [cit. 20. dubna 2017]. 2015. URL: https://www.amazon.com/Learning-WAMP-Server-Development-Online/dp/B00TRA815K/ref=sr_1_3.

[52] SANS Institute InfoSec Reading Room. *An Ettercap Primer*. [online]. [cit. 20. dubna 2017]. 2004. URL: https://www.sans.org/reading-room/whitepapers/tools/ettercap-primer-1406.

[53] Spencer, R. H. a Johnston, R. P. *Technology Best Practices (Wiley Best Practices)*. 1. vyd. Wiley, 2002.

[54] Stallings, W. *Operating Systems: Internals and Design Principles*. 1. vyd. Anybook Ltd., 2004.

[55] Stevens, M. et al. *Announcing the first SHA1 collision*. [online]. [cit. 20. dubna 2017]. 2017. URL: https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html.

[56] Sushil, J. a Chandan, M. *Information Systems Security*. 1. vyd. Springer, 2005.

[57] Syngress Publishing, Inc. *Scene of the Cybercrime: Computer Forensics Handbook*. 1. vyd. Syngress, 2002.

[58] Tanenbaum, A. S. *Computer Networks*. 5. vyd. Pearson India, 2013.

[59] Techopedia Inc. *Proof of Concept (POC)*. [online]. [cit. 20. dubna 2017]. 2017. URL: https://www.techopedia.com/definition/4066/proof-of-concept-poc.

[60] Tilborg, Henk C. A. van a Sushil, J. *Encyclopedia of Cryptography and Security*. 1. vyd. Springer US, 2011.

[61] United States Department of Commerce. *Common Vulnerability Scoring System Calculator Version 3*. [online]. [cit. 20. dubna 2017]. 2017. URL: https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[62] Vostokov, D. *Software Trace and Memory Dump Analysis: Patterns, Tools, Processes and Best Practices*. 1. vyd. Opentask, 2014.

[63] Wilder, F. *Guide to the TCP/IP Protocol Suite*. 2. vyd. Artech House, 1998.

[64] Zahradnický, T. *Data input, Canonical Representation, and Security*. [online]. [cit. 20. dubna 2017]. 2013. URL: https://edux.fit.cvut.cz/courses/MI-BPR.16/_media/lectures/mi-bpr-2014-lecture-8.pdf.

[65] Zahradnický, T. *Introduction into Reverse Engineering*. [online]. [cit. 20. dubna 2017]. 2016. URL: https://edux.fit.cvut.cz/courses/MI-REV.16/_media/lectures/rev-1e.pdf.

[66] Zahradnický, T. *Running with Least Privileges, Windows User Account Control*. [online]. [cit. 20. dubna 2017]. 2013. URL: https://edux.fit.cvut.cz/courses/MI-BPR.16/_media/lectures/mi-bpr-2014-lecture-6.pdf.

[67] Zahradnický, T. *Security Layers, Tokens, and ACLs*. [online]. [cit. 20. dubna 2017]. 2013. URL: https://edux.fit.cvut.cz/courses/MI-BPR.16/_media/lectures/mi-bpr-2014-lecture-5.pdf.

[68] Zetter, K. *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon*. 1. vyd. Broadway Books, 2015.

Příloha A

Seznam použitých zkratek

ACE	Access Control Entry
ACL	Access Control List
API	Application Programming Interface
ARP	Address Resolution Protocol
ASLR	Address Space Layout Randomization
COM	Component Object Model
CRC	Cyclic Redundancy Check
CSV	Comma Separated Values
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
CWSS	Common Weakness Scoring System
DACL	Discretionary Access Control List
DDOS	Distributed Denial Of Service
DEP	Data Execution Prevention
DLL	Dynamic-link Library
DNS	Domain Name System
DNSSEC	Domain Name System Security Extensions
FAT	File Allocation Table
GPL	General Public License
GUI	Graphical User Interface
HKEY	Handle To Registry Key
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secured
IDS	Intrusion Detection System
IP	Internet Protocol
IPC	Inter-process communication
ISP	Internet Service Provider
MAC	Media Access Control
MITM	Man-in-the-Middle
MSVC	Microsoft Visual C++
NTFS	New Technology File System
LGPL	Lesser General Public License
OS	Operating System
OSI	Open Systems Interconnection

PE	Portable Executable
PID	Process Identifier
RPC	Remote Procedure Call
SACL	System Access Control List
SD	Security Descriptor
SEH	Structure Exception Handler
SFX	Self-extracting Archive
SHA	Secure Hash Algorithm
SID	Security Identifier
SO	Securable Object
SSH	Secure Shell
TOR	The Onion Router
UAC	User Account Control
UDP	User Datagram Protocol
USB	Universal Serial Bus
UTF-8	Unicode Transformation Format – 8bit
XML	Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `img` ..... adresář s obrázky
  - `tex` ..... adresář s jednotlivými kapitolami ve zdrojové formě
  - `DP_TRUSINA_Jan_2017.tex` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_TRUSINA_Jan_2017.pdf` ..... text práce ve formátu PDF