VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

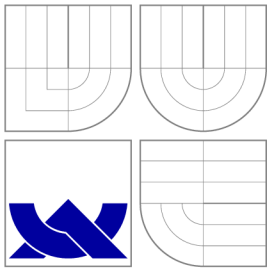
ANALÝZA PARALELIZOVANÝCH NÁSTROJŮ TYPU HONEYPOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

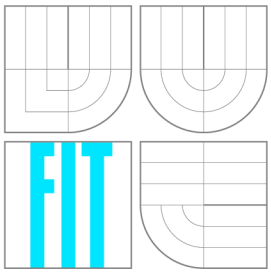
AUTOR PRÁCE
AUTHOR

LUKÁŠ ANTAL

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ANALÝZA PARALELIZOVANÝCH NÁSTROJŮ TYPU HONEYPOT

ANALYSIS OF PARALLEL HONEYPOT TOOLS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ ANTAL
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL DROZD
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá analýzou vybraného nástroje typu shadow honeypot. V práci je vysvětlena potřeba míti k dispozici prostředek pro včasnou detekci nového typu kyberútoku. Analyzovaným nástrojem je software Argos, který je jedním z výsledků práce mezinárodního projektu European Network of Affined Honeypots (NoAH). V práci je provedena jeho důkladná analýza a otestování. Součástí práce je také vytvoření aplikace zpracovávající logy generované nástrojem Argos.

Abstract

This bachelor thesis analyzes the selected shadow honeypot tool. The thesis explaines the need for having tool for early detection of a new type of cyber-attack. Shadow honeypot tool analyzed in the thesis is called Argos. Argos is one of the results of the international project called European Network of Affined honeypots (NoAH). The thesis includes thorough analysis and testing of Argos tool. The paper also includes implementation of Argos log files parsing utility.

Klíčová slova

Honeypot, malware, Conficker, Microsoft-DS, smb, virtualizace, bezpečnost, kyberútok, automatizace, Argos, QEMU

Keywords

Honeypot, malware, Conficker, Microsoft-DS, smb, virtualization, security, cyber-attack, automation, Argos, QEMU

Citace

Lukáš Antal: Analýza paralelizovaných nástrojů typu honeypot, bakalářská práce, Brno, FIT VUT v Brně, 2010

Analýza paralelizovaných nástrojů typu honeypot

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Drozda

.....
Lukáš Antal
11. května 2010

Poděkování

Rád bych poděkoval vedoucímu mé práce panu Ing. Michalu Drozdovi za odborné vedení a čas věnovaný konzultacím. Dále bych rád poděkoval panu Ing. Petru Chmelařovi za poskytnuté rady a připomínky. Děkuji také své přítelkyni za trpělivost a podporu.

Obsah

Kapitola 1

Úvod

V posledních letech[1] rapidně narůstá počet případů kyberútoků za použití počítačových virů, trojských koní, spyware a jiných typů malware[2]. Tyto útoky mohou kompromitovat rozsáhlé sítě během několika málo minut. Takový průběh útoku je pak příliš rychlý na to, aby zodpovědní lidé mohli včas reagovat. Proto je nadmíru potřebné detekci útoků a jiného podezřelého chování automatizovat. Útok je potřebné zachytit včas a přijmout odpovídající opatření.

Mnoho uživatelů počítačů žije v domnění, že nainstalováním dobrého antivirového systému jsou před malware v bezpečí. Antivirové programy však spoléhají hlavně na svoji databázi vzorků známých typů malware. Naprosto nový dosud nezdokumentovaný typ malware tak tímto filtrem lehce projde. Antivirové programy dále používají heuristiky k rozpoznání škodlivého kódu. Ty jdou bohužel vcelku snadno oklamat polymorfizací kódů, tedy procesem, který co nejvíce změní vnitřní strukturu souboru, přičemž zachová veškeré funkce programu. Detekce takového polymorfního malware je pak mnohem náročnější[2].

To, jak je svět na hrozbu malware nepřipraven, dokázal koncem roku 2008 počítačový virus *Conficker*[3]. *Conficker* sám o sobě nepřišel s ničím novým, pouze efektivně zkombinoval výhodné vlastnosti jiných typů malware. Původní verze viru využívala zranitelnost systému Microsoft Windows, která se vyskytovala ve službě pro sdílení zdrojů zvané *Samba*. Označení této zranitelnosti je **MS08-67**[4]. Rozšířenost této chyby napříč několika verzemi systému Windows umožnila viru velice rychlé šíření. V pozdější verzi byla viru přidána schopnost šířit se za pomocí bootovatelných médií. O tom, kolik celkem Conficker napadl počítačů, se dodnes pouze spekuluje. Různé oficiální výzkumy přinášejí rozdílná čísla. Jistotou však je, že je toto číslo v řádu milionů unikátních IP adres.

Pryč je také doba, kdy psaní počítačových virů a jiných typů malware bylo doménou jednotlivců, chtivých škodit či s touhou se zviditelnit v rámci své komunity. V současné době je psaní a distribuce malware promyšleným byznysem, do kterého jsou zapojeny dobře organizované skupiny[5]. Touhu dosáhnout uznání svých kolegů tak nahradila motivace jediná a to peníze.

[1] http://www.pandasecurity.com/img/enc/Annual_Report_PandaLabs_2009.pdf
[2] Pojem malware bude blíže vysvětlen v kapitole 2.
[3] Confickeru je věnována sekce 2.3.
[4] http://www.microsoft.com/technet/security/Bulletin/MS08-067.mspx
[5] http://en.wikipedia.org/wiki/Russian_Business_Network

1.1 Cíle práce

V této práci bude vysvětlena potřeba míti systém na včasnou detekci kyberútoků. Z důvodu masivního nárůstu výskytu stále nových a nových typů malware je stále obtížnější nové vzorky zaznamenávat, následně tvořit signatury těchto útoků a ty distribuovat do nástrojů bojujících proti této hrozbě. Doba mezi vypuštěním nového typu malware a jeho detekcí antimalwarovými produkty se stále prodlužuje. Z tohoto důvodu je výhodné činnost sběru dat z útoků a jejich zařazování do databáze a distribuci co nejvíce automatizovat. Výsledkem by tak měla být architektura okamžitě reagující na nový útok. Jakmile by takový útok proběhl, byl by okamžitě zaznamenán a vytvořené signatury okamžitě poskytnuty jako aktualizace do anti-malware nástrojů. Vše by ideálně mělo proběhnut v řádů maximálně jednotek vteřin. Srdcem takovéto architektury pak bude nástroj typu shadow honeypot. Nástroje tohoto typy využívají ty nejlepší vlastnosti ze systémů honeypot a Anomaly Based Detection systémů. Ve výsledku tak postytují vysokou míru přesnosti detekce a jsou schopny rozpoznat širokou škálu útoků. Tato práce se zabývá analýzou právě takového nástroje, sloužícího pro sběr dat z kyberútoků. Myšlenka koncepce shadow honeypot je poměrně čerstvá a tak není k dispozici mnoho funkčních implementací těchto nástrojů. Mým úkolem je tak zhodnocení současného stavu této oblasti a výběr vhodné implmentace. Po důkladné analýze nástroje navrhnu a implementuji řešení automatizující výše zmíněný proces sběru a ukládání dat z útoků.

V kapitole 2 bude popsán pojem malware, jeho rozdělení do různých kategorií, vývoj malwaru v posledních letech a jako jeden v současnosti z nejznámějších vzorků malware bude popsán *Conficker*. V téže kapitole bude také vysvětlen v současné době hodně skloňovaný a s malwarem úzce spojený pojem *botnet*. Na konci kapitoly budou stručně přiblíženy hlavní postupy, které se při analýze škodlivého softwaru uplatňují.

V kapitole 3.1 budou vysvětleny pojmy *Intrusion Detection System* a *Intrusion Prevention System*. Dále zde budou uvedeny rozdíly mezi těmito systémy a popsáno jejich rozdělení do skupin podle určitých kategorií.

Od IDS a IPS je to už jen krůček k systémům typu *honeypot*. Tomuto pojmu, který je pro tuto práci klíčový, se věnuje kapitola 3.4. Stejně jako IDS a IPS budou i honeypoty rozděleny do určitých kategorií. Mírně stranou pak stojí pojem *shadow honeypot*[6], který bude objasněn v téže kapitole.

V kapitole 4 bude představen projekt European Network of Affined Honeypots neboli NoAH. Dále budou v této kapitole představeny výsledky tohoto projektu, hlavně výsledné implementace týkající se honeypotů. Z významných implementací to budou *honey@home* (5) a nástroj *Argos* (6).

Nástroji Argos pak bude věnována celá kapitola 6. Aplikace budou nejdříve popsána a analyzována a následně i důkladně otestována. Důležitým výstupem útoku jsou log soubory generované tímto nástrojem. Jelikož jedním z hlavním cílů v boji s malware je automatizace tohoto procesu, bude součástí práce také implementace aplikace provádějící zpracování záznamů z útoků a jejich následné zařazení do databáze. Popisu a implementaci této aplikace bude věnována kapitola 7.

Výsledky a výstupy práce a možná další možná rozšíření budou shrnuty v závěrečné kapitole 8.

[6] Dále bude používáno také označení **paralelizovaný honeypot** se stejným významem.

Kapitola 2

Malware a jeho detekce

Hlavní činnost vybrané implementace nástroje typu shadow honeypot tkví ve sběru dat z kyberútoků. V této kapitole bude popsán hlavní původce automatizovaných útoků, jímž je skupina škodlivých nástrojů obecně označovaných jako *malware*.

2.1 Typy malware

Slovo malware vzniklo složením anglických slov *malicious software*. Tímto termínem se označuje veškerý škodlivý kód, který slouží pro neoprávněné získání přístupu, krádež či poškození dat. V dnešní době je však obtížné konkrétní malware přiřadit do určité kategorie, jelikož zpravidla splňuje charakteristiky více skupin. Přesto zde stručně popíši charakteristiky jednotlivých skupin malwaru.

Trojské koně

Jako trojský kůň označujeme aplikaci, jež pod záminkou určité funkcionality nainstaluje do systému škodlivý kód.

Počítačové viry

Hlavní schopností viru je šíření sama sebe bez přispění uživatele. Virus infikuje další soubory, do kterých pak zkopíruje svůj kód. Cílovými soubory, které virus infikuje, bývají ve většině případů spustitelné soubory.

Počítačoví červi

Definice počítačového červa se z velké části překrývá s definicí počítačového viru. Jedná se o software, jež se bez vědomí uživatele rozesílá na jiné stroje. K šíření používá různé komunikační kanály jako například e-maily, ICQ, IRC a další. Hlavním rozdílem oproti viru je pak skutečnost, že červ nepotřebuje další soubory k tomu, aby se mohl šířit.

Spyware

Spyware je typem malware, který odesílá informace z počítače bez vědomí uživatele. Takovou informací může být popis chování uživatele na Internetu pro potřeby reklamy jako

například seznam navštívených www stránek či hledané fráze ve vyhledávači. V horším případě pak například odesílání údajů o kreditní kartě či přístupové údaje k citlivým službám jako je třeba Internetové bankovnictví.

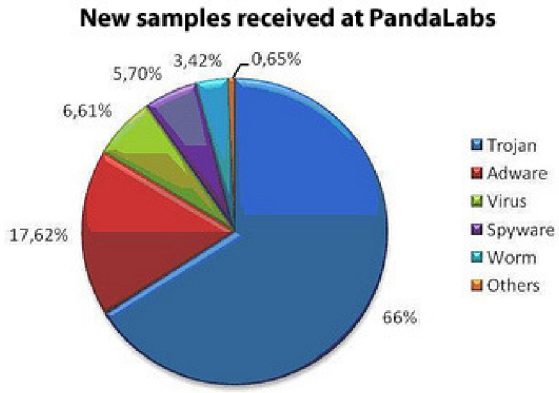
Rootkity

Pojem rootkit původem pochází z unixového světa, což tvoří výjimku oproti ostatním typům malware, které jsou v drtivé většině případů určeny již od jejich počátků pro platformu Windows. Samotný rootkit je software, který maskuje svoji přítomnost případně přítomnost ostatních typů malwaru (viry, trojské koně, ...) na daném počítači. Kvalitní rootkit jde pak na daném stroji velice těžce odhalit.

2.2 Vývoj v posledních letech

Rok 2009 byl z hlediska malware přelomovým. Za rok 2009 bylo zaznamenáno více nových vzorků malwaru než v součtu dohromady za posledních dvacet let[9]. V roce 2010 se předpovídá podobný skokový nárůst. Ke konci roku 2009 zaznamenávaly scannery společnosti PandaLabs[1] více než 50000 nových vzorků denně. Nejen toto svědčí o tom, jaká lavina se ze strany malwaru hrne na počítače připojené k Internetu.

Produkce malware už dlouho není doménou jednotlivců, snažících se na sebe upozornit. V současné době je naopak dílem organizovaných skupin. Jedná se o propracovaný a velice výnosný obchod. Ze všech zachycených vzorků malwaru, které v PandaLabs zachytili, vítězí drtivě trojské koně se 66 %. Následuje adware se 17,62 %, viry s 6,61 %, spyware s 5,70 % a červi se 3,42 %. Zastoupení jednotlivých typů malware je přehledně zobrazuje obrázek 2.1. Pokud bychom dále dělili trojské koně podle určení, vedoucí pozici co do rozšířenosti by obsadili bankovní trojské koně. To není vůbec překvapující, už jen z důvodu, že obchod s bankovními údaji je v šedé zóně Internetu velmi rozšířený.



Obrázek 2.1: Procentuální zastoupení různých typů malware ve vzorcích zachycených za rok 2009. Převzato z [9]

[1] http://pandalabs.pandasecurity.com

2.3 Conficker

Pokud by se měl zvolit malware s největším mediálním ohlasem za poslední rok, byl by to bezpochyby malware *Conficker*. Přestože se začal šířit již na konci roku 2008, je o něm stále slyšet. Jméno Conficker není jediný název pro tento malware, je však rozhodně nejpoužívanější a je tak označován antivirovými společnostmi CA, McAfee, Eset, Panda a dalšími. Dalšími názvy jsou například *Downadup* (F-Secure, Symantec) či *Kido* (Kaspersky). Pojmenování[2] Confickeru několika nejznámějšími společnosti zabývajícími se bojem s malwarem je shrnuto v tabulce 2.1. Z ní jasně vyplývá, že označení Conficker je nejrozšířenější.

2.3.1 Původ názvu

Jak vlastně tento název vznikl? Teorií je více, avšak nejpravděpodobnější se jeví, že Conficker je složenina slov *Configure* a *Ficken*, jež je německým ekvivalentem anglického vulgarismu *fucker*. Tento název se vztahuje ke skutečnosti, že malware pozmění konfiguraci počítače, což spočívá ve vypnutí automatických aktualizací systémů, přepsání DNS záznamů aktualizačních serverů a deaktivování antivirových a antimalwarových programů[6].

Antivirus	Označení
Avast	Win32:CoPack
AVG	Worm/Generic.WLO
BitDefender	Win32.Worm.Downadup.Gen
Comodo	Worm.Win32.Exploit.Conficker
F-Secure	Worm:W32/Downadup
Kaspersky	Trojan-Downloader.Win32.Kido
McAfee	W32/Conficker.worm.gen
NOD32	Win32/Conficker
Norman	W32/Conficker
Panda	W32/Conficker
Sunbelt	Worm.Win32.Downad.Gen
Symantec	W32.Downadup

Tabulka 2.1: Přehled názvů pro Conficker používaných jednotlivými antiviry.

[2]Názvy převzaty z http://www.virustotal.com

2.3.2 Šíření

Conficker je zajímavý tím, že sám o sobě nepřišel s žádnou přelomovou vlastností, pouze vhodně zkombinoval vlastnosti již existujících druhů malwaru. První verze využívaly zranitelnost systému Windows v síťové službě sdílení souborů a tiskáren. Tato zranitelnost, samotným Microsoftem označena jako kritická[3], se týkala systémů **Windows 2000**, **Windows XP** a **Windows Server 2003** a v omezené míře i systémů **Windows Vista** a **Windows Server 2008**. Přestože Microsoft uvolnil záplatu již 23. října 2008, tedy na samém počátku epidemie, měl *Conficker* poměrně dost času na to se masově rozšířit. Důvodem byl vedle velkého rozsahu postižených verzí systémů Windows i ignorantský přístup mnoha uživatelů k aktualizacím systému.

Po infikování počítače malware aktivně scanuje okolní síť a snaží se najít další zranitelné počítače. Kromě zneužití zranitelnosti **MS08-067** se Conficker dále šíří slovníkovým útokem na sdílené zdroje v LAN sítích a přes automatické spouštění obsahu odpojitelných médií.

Epidemie, kterou *Conficker* způsobil, je jednou z největších za posledních několik let. Počty nakažených počítačů se velmi těžko odhadují, jisté však je, že se toto číslo pohybuje v řádu milionů[4]. Výzkumy na odhad počtu nakažených PC jsou založeny na scanování unikátních IP adres v Internetu. Výsledné číslo však bude ještě mnohem vyšší v důsledku infikace celých podnikových i soukromých LAN sítí.

2.3.3 Funkce

Infikovaný počítač je ihned zapojen do *botnetu*. Ten slouží k rozesílání nevyžádané pošty, útokům distribuovaného odmítnutí služby či krádeži osobních údajů. Dále botnet slouží k dalšímu šíření malwaru, tedy každý infikovaný počítač začne aktivně vyhledávat další zranitelné cíle, což způsobuje lavinový efekt šíření.

2.3.4 Verze

První verze červa *Confickeru* se šířila pouze zneužitím zranitelnosti **MS08-067**. Další verze tohoto malwaru přidaly nové, již jmenované způsoby šíření a to přes automatické spouštění obsahu odpojitelných médií či přes sdílené složky v LAN sítích. Dosud bylo popsáno pět verzí tohoto malware. Tyto verze se označují písmeny A až E. Data[5] prvního výskytu každé verze jsou uvedena v tabulce 2.2.

2.3.5 Dopad ve světě

O červu *Conficker* se toho namluvilo a napsalo mnohem více než o jakémkoliv jiném malware z poslední doby. Důvodem je mimo jiné i dopad v reálném světě, jakým bylo například uzemnění stíhacích letounů francouzského námořnictva. Počátkem roku 2009 nemohly letouny odstartovat z důvodu nemožnosti stažení letových plánů z databáze[6]. Interní síť námořnictva totiž byla offline z důvodu infekce červem a námořnictvo se tak muselo na nějaký čas vrátit ke komunikačním prostředkům jako jsou telefony a faxy.

[3] MS08-067: http://www.microsoft.com/technet/security/Bulletin/MS08-067.mspx

[4] http://www.upi.com/Top_News/2009/01/26/Virus-strikes-15-million-PCs/UPI-19421232924206/

[5] Čerpáno z http://www.microsoft.com/security/worms/Conficker.aspx#EWC [27.1.2010].

[6] http://www.telegraph.co.uk/news/worldnews/europe/france/4547649/French-fighter-planes-grounded-by-computer-virus.html

Verze	Datum prvního výskytu
Win32/Conficker.A	21. listopadu 2008
Win32/Conficker.B	29. prosince 2008
Win32/Conficker.C	20. února 2009
Win32/Conficker.D	4. března 2009
Win32/Conficker.E	8. dubna 2009

Tabulka 2.2: Data prvních výskytů jednotlivých verzí malwaru Conficker.

V červnu stejného roku nemohl magistrát města Manchester rozdat pokuty 1609 řidičům zachyceným kamerami z důvodu nemožnosti odesílat emailové zprávy a tisknout dokumenty. Výsledná škoda činila 1,5 milionu liber[7]. Viníkem byl opět Conficker.

Conficker si nevybíral a infekci tak dále podlehla například počítačová síť nemocnice v Sheffieldu, síť ministerstva obrany Velké Británie, síť německé armády Bundeswehr a mnohé další. Dříve byly podobné akce spojené s počítačovými viry spíše doménou scifi filmů. S příchodem moderního malware se však stávají skutečností.

Autoři tohoto malwaru zůstávají stále neznámí, přestože již počátkem roku 2009 vypsal Microsoft odměnu 5 milionů amerických dolarů za jejich dopadení[8].

2.4 Analýza malware

Samotné rozpoznání malwaru a jeho odlišení od legitimních aplikací tvoří jádro antivirových a jiných antimalwarových nástrojů. Na tuto problematiku se v zásadě uplatňují základní postupy. První, starší a jednodušší metodou je rozlišování malwaru pomocí tzv. otisků (signatures). Druhou, mnohem složitější metodou je analýza samotného souboru. Ta se dále může dělit na statickou analýzu a analýzu chování.

2.4.1 Podle otisku

Tato metoda funguje na principu, že z každého nového nalezeného vzorku malware je vytvořen tzv. otisk. Všechny tyto otisky se ukládají do databáze. Při kontrole testovaného souboru je proveden otisk tohoto souboru a porovnáván s otisky různých typů malware uložených v databázi. Pokud se otisk testovaného programu shoduje s některým otiskem z databáze, je testovaný soubor prohlášen za daný typ malware a jsou dále podniknuty kroky nastavené výrobcem antimalwarového produktu.

V dobách, kdy bylo unikátních vzorků malwaru poměrně málo a frekvence výskytu nových typů byla nízká, byla tato metoda poměrně efektivní. Se zvýšenou frekvencí výskytu nového malwaru však roste náročnost sběru otisků a jejich zařazování do databáze. Tím

[7] http://www.theregister.co.uk/2009/07/01/conficker_council_infection/

[8] http://ipcommunications.tmcnet.com/topics/ip-communications/articles/50562-microsofts-5000000-reward-the-conficker-worm-creators.htm

se prodlužuje čas, který uplyne od doby, kdy je malware vypuštěn, do doby, kdy je malware antimalwarovým nástrojem detekován. Tato situace se označuje jako *Signature Arms Race*[11] a výrobci malware z ní těží.

Další skutečností hrající v neprospěch analýzy založené na otisku je možnost malware přetvořit tak, aby měl odlišný otisk a přitom stejnou funkcionalitu. To může být způsobeno například těmito faktory[11]:

- Vytvoření rutin, které zašifrují kód pomocí silných kryptografických mechanismů a náhodných klíčů.
- Přetvoření celého základu kódu pomocí polymorfizujících funkcí
- Zabalení a komprese spustitelných souborů

Co se třetí metody týče, výzkum PandaLab z roku 2007 odhalil, že 78% malware používá tuto techniku[9] s tím, že nejpopulárnějším nástrojem pro zabalení a kompresi je open-source aplikace **UPX**[10], která se na špici pevně drží již několik let. Proti těmto způsobům změny otisku už zůstává analýza založené na otisku bezmocná.

Do karet útočníkům také hrají bezplatné služby typu *VirusTotal.com*, kde si výrobci malware rychle a efektivně ověří, jak jejich nový kousek obstojí proti přibližně čtyřicítce známých antivirových nástrojů. Sám jsem tuto služby vyzkoušel na vzorku malware Conficker. Všech 41 antivirových systémů Conficker odhalilo[11]. Poté jsem na soubor infikovaný Confickerem použil polymorfní kodér *Shikata Ga Nai*. Ani jeden antivirus poté soubor neohlásil[12], přestože stále skrýval nebezpečný malware.

2.4.2 Na základě analýzy souboru

Do této kategorie spadají tzv. heuristiky. Ty jsou pro jednotlivé výrobci antimalwarových řešení dobře střežená obchodní tajemství, jelikož v nich většinou tkví srdce a mozek samotného nástroje. Heuristiky spočívají v následujících dvou oblastech:

Statická analýza

Při této analýze testovaného souboru se soubor nespouští, ale probíhá jeho disassembling, tedy převod do jazyka symbolických instrukcí. V případě, že tento úkon provádí automatizovaný nástroj, není disassembling nutný. V souboru jsou pak hledány různé vzory, které mohou nasvědčovat o škodlivosti či naopak neškodnosti kódu.

Behaviorání analýza

V této fázi analýzy dochází ke spouštění analyzovaného programu. Spuštění neprobíhá na produkčním systému, ale na virtuálním stroji či v jiném uzavřeném prostředí (tzv. Sandbox). Po spuštění programu je monitorována jeho aktivita, zápis na disk, síťová komunikace a další parametry, na základě kterých se rozhodne o tom, zda program vykonává škodlivé funkce či nikoliv.

[9] http://www.pandasecurity.com/homeusers/media/press-releases/viewnews?noticia=8612
[10] http://upx.sourceforge.net/
[11] http://bit.ly/9BGaLP
[12] http://bit.ly/djlbNH

Kapitola 3

Honeypot a IDS/IPS

Jak již z názvu práce vyplývá, tato práce se zabývá analýzou nástroje typu honeypot, kterým bude věnována tato kapitola. Před vlastním vysvětlením tohoto pojmu však budou ještě popsány pojmy IDS a IPS, které s tématem taktéž úzce souvisejí. Ve výsledku totiž data nasbíraná pomocí honeypotu budou přetransformována do aktualizací a pravidel právě do nástrojů IDS/IPS. Ty drží první linii v boji proti malware a jejich aktualizace kvůli rozpoznávání těch nejnovějších hrozeb je kriticky důležitá.

3.1 IDS

IDS (Intrusion Detection System) je nástroj sloužící k detekci podezřelého chování v určitém systému či v počítačové síti. Podle této definice můžeme IDS rozdělit do dvou skupin a to právě podle oblasti, v níž nástroj detekci provádí.

3.1.1 Hostitelské IDS (Host IDS, HIDS)

Hostitelské IDS bývá zpravidla aplikace nainstalovaná přímo na koncový systém, jehož lokální aktivity jsou monitorovány. Jedná se například o modifikace souborového systému, změny v registrech (u MS Windows), analýza log[1] souborů aplikací či monitorování systémových volání.

3.1.2 Síťové IDS (Network IDS, NIDS)

IDS tohoto typu mohou být jak softwarová tak hardwarová. Síťové IDS na rozdíl od předchozího typu nedohlížejí pouze na jedno zařízení, ale monitorují veškerou komunikaci, která přes něj proudí. IDS tohoto typu tak musí být napojeno na některý uzlový bod sítě jako hub, switch či router. Principem NIDS je analýza datového provozu. Ten je na základě určitých pravidel vyhodnocován a veškeré podezřelé aktivity jsou zaznamenány.

3.1.3 Vyhodnocování provozu

S vyhodnocováním provozu a rozpoznáváním podezřelých aktivit souvisí pojmy *True Positive, False Positive, True Negative, False Negative*. Tyto pojmy jsou v oblasti IT bezpečnosti již zažité a nepřekládají se. Význam pojmů je shrnut v tabulce 3.1.

[1] Spojení *log soubor* bude ve významu soubor obsahující záznam o průběhu činnosti programu či se záznamem určité akce používáno po celý zbytek práce.

True Positive	Na daný systém byl veden útok a IDS ho odhalil a ohlásil.
False Positive	Na daný systém nebyl veden žádný útok, IDS však útok oznámil.
True Negative	Na daný systém nebyl veden žádný útok, IDS nic neohlásil.
False Negative	Na daný systém byl veden útok, IDS ho však neohlásil.

Tabulka 3.1: Význam pojmů v souvislosti s IDS systémy

V ideálním případě by IDS měl generovat pouze samá *True Positive* hlášení. Tato situace je však velice těžce realizovatelná. V praxi IDS generuje určité procento *False Positive* i *False Negative* hlášení. Pokud převažují *False Positive* hlášení, má systém nastavena příliš přísná kritéria vyhodnocování. Naopak příčinou zvýšeného počtu *False Negative* hlášení může být příliš benevolentní nastavení ohodnocovacích pravidel. Základem je IDS vyladit tak, aby byl poměr mezi *False Negative* a *False Positive* přibližně stejný a v celkovém měřítku co nejnižší.

3.2 ADS

Klasické IDS systémy používají pro zaregistrování útoku určitou sadu pravidel či signatur. Toto je velké omezení, které určuje, že takový IDS systém je pak schopný zachytit pouze útok, který byl již někdy v minulosti zaznamenán. Proti tzv. zero-day útokům, tedy útokům, které dosud nebyly zaznamenány, jsou pak bezbranné. Zde nastupují tzv. **Anomaly-based intrusion detection system**, zkráceně **Anomaly detection system** (ADS). ADS jsou novou generací IDS systémů, které k detekci útoků používají heuristiky či určité metody z oboru umělé inteligence a neuronových sítí.

3.3 IPS

IPS (Intrusion Prevention System) je nástroj charakteristikou podobný IDS. IPS však kromě monitorování provozu na podezřelé aktivity i určitým způsobem reaguje. Takže zatímco IDS útok pouze ohlásí a o více se nestará, IPS může navíc probíhající útok zablokovat či přesměrovat na jiný segment sítě, kde se například vyskytují honeypoty (viz kapitola 3.4) s tím, že ostatní komunikaci nechá nedotčenou. V praxi se však IPS a IDS příliš nerozlišují a používají se jejich kombinace, tzn. Intrusion Prevention/Detection system (IPDS), které dokáží útok jak odhalit, tak mu více či méně zabránit.

3.4 Honeypot

Název honeypot by šel přeložit jako hrnec s medem, avšak v praxi se tento termín nepřekládá. Honeypot je nástroj, jehož účelem je přilákat pozornost narušitele v síti, ať už je to člověk nebo automatizovaný nástroj. Honeypot může být software, který emuluje jedinou službu či rovnou celý počítač nebo to může být přímo fyzický počítač v síti. Jelikož útočník se ve většině případů nejprve zaměří na nejslaběji zabezpečený článek sítě, je velmi

pravděpodobné, že se pokusí honeypot kompromitovat. Vlastník honeypotu tak na rozdíl od útočníka získá cenná data. Tato data zahrnují provedení útoku a škodlivý kód odeslaný na server. Tyto údaje jsou poté zpracovány a slouží jako základ vytvoření prevence proti danému typu útoku.

Honeypot zpravidla emuluje služby, u kterých se očekává, že budou útočníkem napadeny. Seznam takových služeb shrnuje tabulka 3.2. Hlavním důvodem nasazení honeypotů je potřeba zjistit slabiny dané emulované služby.

Protokol	Použití	Port
FTP	Přenos souborů	21
SSH	Bezpečný vzdálený přístup	22
TELNET	Vzdálený přístup	23
SMTP	Odesílaní pošty	25
HTTP	Prohlížení webových stránek	80
POP3	Příjímání pošty	110

Tabulka 3.2: Standardní služby emulované honeypoty

Honeypoty můžeme dělit podle různých kritérií.

3.5 Dělení honeypotů

Honeypoty můžeme dělit podle toho, do jaké hloubky emulují danou službu a co vše umožňují útočníkovi provádět na:

1. Honeypoty s nízkou úrovní interakce

Do této kategorie spadá většina dnešních honeypotů. Ty nesimulují danou službu či služby v jejich plném rozsahu. Jejich cílem je hlavně nalákat útočníka a zachytit síťovou komunikaci, která je poté podrobena analýze. Zkušenější útočník tak brzy pozná, že se jedná o honeypot.

2. Honeypoty s vysokou úrovní interakce

Honeypot s vysokou úrovní interakce je koncipován tak, aby útočníkovi dovolil systém zcela kompromitovat. Díky tomu majitel honeypotu zjistí oproti předchozímu typu honeypotů nejen to, jak je proveden pokus o kompromitaci systému, ale i následné útočníkovi kroky a jeho záměry.

Dále honeypoty dělíme podle toho, zda se jedná o softwarový nástroj či přímo celý vyhrazený fyzický počítač.

1. Virtuální systémy typu honeypot

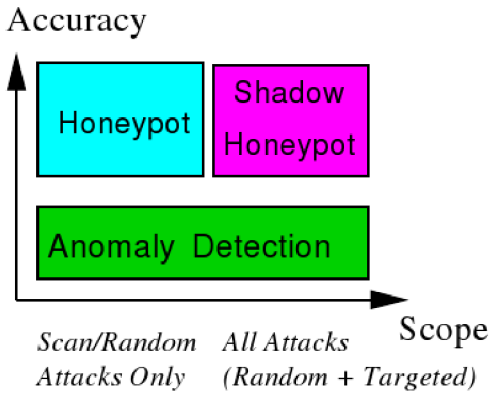
Virtuální honeypot je software, který umožňuje emulovat službu nebo služby či rovnou celý operační systém. Výhodou tohoto řešení je cena, jelikož není nutné pro honeypot vyhrazovat vlastní hardware. Naopak na jednom fyzickém serveru může běžet mnoho takových honeypotů.

2. Fyzické systémy typu honeypot

Jsou vyhrazené fyzické počítače umístěné v rámci sítě s vlastní IP adresou. Na těchto počítačích potom běží software podporující simulaci cílové služby. Nevýhodou tohoto řešení je vyšší cena oproti předchozímu typu.

3.6 Shadow honeypot

Pojem Shadow honeypot označuje novou hybridní architekturu, která kombinuje to nejlepší z vlastností standardních honeypotů a ADS systémů. Z ADS systémů shadow honeypot přebírá rozsah možných útoků, které je schopen zajistit, zatímco funkce honeypotu mu dává vysokou přesnost detekce. Tento vztah popisuje obrázek 3.1.



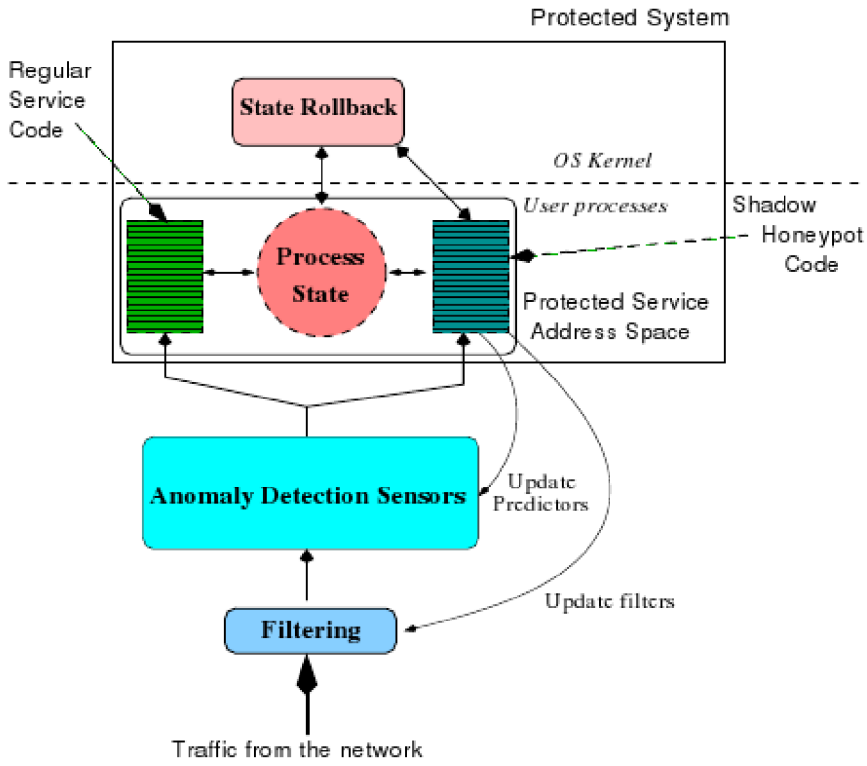
Obrázek 3.1: Klasifikace pojmu shadow honeypot. Převzato z [1]

Shadow honeypot, který je umístěn v chráněné zóně sítě, sdílí interní stav s určitou produkční aplikací. Pokud je na tuto službu veden útok, je pomocí mechanizmů implementovaných v shadow honeypotu zastaven a zaznamenán. Informace o útoku jsou poté předány IDS/IPS/ADS systémům v téže síti, které zajistí, aby byl daný útok příště okamžitě zablokován. V případě, že aplikaci dojdou validní data, jsou shadow honeypotem neovlivněny a aplikace je tak může zpracovat.[1].

Tato koncepce se tak výrazně liší od klasického pojetí pojmu honeypot, kdy standardní honeypot danou službu pouze předstírá, aby nalákal útočníka, zatímco aplikace chráněná shadow honeypotem v případě validních dat danou službu opravdu poskytuje. Hlavní výhodou shadow honeypotu oproti klasickým IDS/IPS systémům je téměř nulové generování hlášení typu *false positive* (viz tabulka 3.1).

Architektura shadow honeypotu je zachycena na obrázku 3.2. Z obrázku je názorná spolupráce shadow honeypotu s ADS systémem, jehož heuristické metody jsou po každém

novém (*zero-day*) útoku vedeném na danou aplikaci aktualizované daty ze zachyceného útoku.



Obrázek 3.2: Architektura shadow honeypotu. Převzato z [1]

3.7 Shrnutí

Většina IDS a IPS nástrojů nejde klasifikovat jako čistá IDS nebo IPS. Jedná se většinou o kombinace IDS/IPS nebo IDS/Honeypot. Příkladem používaných aplikací může být nástroj **LaBrea**[2], který kombinuje vlastnosti IDS a honeypotu či nástroj **Snort**[3], který zase kombinuje IDS a IPS. Příkladem klasického honeypotu je pak aplikace **honeyd**[4].

Trochu stranou od těchto pojmů pak stojí zcela nový koncept zvaný shadow honeypot, který kombinuje ty nejlepší vlastnosti nástrojů typu honeypot (přesnost detekce útoku) a Anomaly Detection systémů (široké spektrum typů detekovaných útoků). Analýza nástroje tohoto typu je pak náplní dalších částí této práce.

[2] http://labrea.sourceforge.net/labrea-info.html
[3] http://www.snort.org/
[4] http://www.honeyd.org

Kapitola 4

Projekt NoAH

V této práci vycházím z výsledků[1] práce projektu European Network of Affined Honeypots[2] (NoAH). Tato kapitola bude pojednávat o tom, o jaký projekt se jednalo a čím se zabýval. Projekt NoAH si dal za cíl shromáždit a analyzovat informace týkající se kyberútoků představovaných především viry, červy a jiným malware na Internetu. Na základě těchto informací se lidé stojící za projektem snažili vytvořit systém na detekci nových útoků a obrany proti nim. Jak už ze samotného názvu projektu vyplývá, je tento systém založen na technologii honeypot.

Projekt započal 1. dubna 2005 a pokračoval až do 31. března 2008. Na projekt bylo vynaloženo celkem 2 429 374 EUR, z čehož se jednalo ze 60% o dotace Evropské unie. Na projektu se podíleli odborníci z Francie, Německa, Nizozemska, Řecka a Švýcarska.

4.1 Cíle projektu NoAH

- Navrhnout moderní architekturu založenou na nástrojích typu honeypot, která bude sbírat a zpracovávat data z kyberútoků.
- Vyvinout způsob automatizace identifikace útoků a automatického generování jejich signatur. Dále vyvinout mechanizmy k distribuci těchto signatur do firewallů a jiných kontrolních systémů.
- Nainstalovat a zprovoznit prvotní honeypot infrastrukturu, na níž bude demonstrována užitečnost a efektivita distribuovaných bezpečnostních systémů. Toto bylo plánováno na poslední rok vývoje.
- Sbírat data z kyberútoků s cílem zkoumat nové trendy, zdokonalit bezpečnostní modely a celkově podpořit obdobné výzkumy na Internetu.
- Zveřejnit výsledky projektu včetně open-source softwaru. Anonymizovaná data z provozu poskytnout různým **NREN** (National research and education network), **ISP** (Internet service provider) a **CSIRT** (Computer Security Incident Response Team).

[1] http://www.fp6-noah.org/downloads/
[2] http://www.fp6-noah.org/

4.2 Hotové implementace

Výsledkem projektu NoAH jsou mnohé odborné publikace na téma honeypot. Mimoto jsou však na stránkách projektu k dispozici i hotové implementace prototypů různých nástrojů. Jedná se přímo o implementace paralelních honeypotů nebo různých pomocných nástrojů. Právě z těchto implementací bylo mým úkolem vybrat jednu nebo více a ty důkladně analyzovat. Následuje stručný přehled dostupných implementací:

Argos Secure System Emulator

Argos je emulátor operačních systémů navržený pro využití jakožto honeypot. Nástroj je nadstavbou nad známým opensource emulačním softwarem *QEMU*[3]. Argos, stejně jako *QEMU*, podporuje množství operačních systémů a typů CPU. Operační systém emulovaný přes Argos nepotřebuje žádné modifikace. Tomuto nástroji se věnuje celá kapitola 6.

honey@home client

honey@home je jedna z klientských implementací projektu NoAH zaměřující se na usnadnění sběru informací z kyberútoků. Nástroj může být nainstalován pod operačním systémem Linux[4] i Microsoft Windows. Navržen je tak, aby byl lehce ovladatelný a nezatěžoval prostředky počítače. Program běží jako proces na pozadí a v případě získání dat z útoku komunikuje s centrálním honeypot serverem. Tomuto nástroji je dále věnována následující kapitola 5.

NOAH Database Population Tool

NOAH Database Population Tool neboli NOAHDB je konzolová aplikace pro zpracování výstupních souborů honeypotů. NOAHDB převádí data z log souborů do databáze a tak pomáhá síťovým administrátorům shromažďovat a analyzovat cenné informace z útoků.

NOAH Database Management Interface

NOAH Database Management Interface neboli NOAHIF je webová aplikace založená na frameworku Ruby on Rails[5], jejímž cílem je usnadnit správu sítě honeypotů. Aplikace umožňuje vložení údajů týkajících se jednotlivých honeypotů jako je například poloha, hardware, softwarová konfigurace či běžící služby.

Shelia client-side honeypot

Shelia je jednoduchý IDS[6] klient určený pro operační systém Windows. Nástroj umožňuje procházet všechny emaily zadané emailové schránky a otevírat přílohy a následovat případné HTML odkazy. Sheila se tak vlastně snaží simulovat chování naivního uživatele emailové schránky. Během toho tato aplikace monitoruje jednotlivé procesy a vyhlásí alarm v případě podezřelé činnosti.

[3] http://www.qemu.org

[4] Přesné označení tohoto operačního systému je GNU/Linux, dále v této práci však budu používat pouze pojmenování Linux ve smyslu operačního systému jako celku, ne pouze jádra systému.

[5] http://rubyonrails.org

[6] Viz. kapitola 3.1.

Signature Generator

NoAH Signature Generator je určen ke kooperaci s jinou aplikací a generování otisku (signatur) útoku. Nástroj je navržen a implementován jako framework s možností využití plug-in modulů a mechanismu šablon. Obsahuje také modul určený k logování a vyvažování zátěže na systémech s vícejádrovými procesory.

4.3 Požadavky na implementaci a její výběr

Právě z výše uvedených nástrojů je mým cílem vybrat několik implementací splňujících co nejvíce požadavků a ty důkladně analyzovat. Požadavky kladenými na aplikaci jsou:

- Nástroj typu honeypot
- Schopnost paralelizace služby[7]
- Otevřený kód
- Rozpoznávání činnosti malwaru na základě chování (viz. 2.4.2)
- Zpracování výsledků do relační databáze

Některé z těchto požadavků jsou velmi konkrétní a je tak pravděpodobné, že dokonce žádná z implementací je nebude splňovat všechny. V seznamu výsledných implementací projektu NoAH jsou celkem tři aplikace fungující jako honeypot. Jsou jimi **Argos**, **honey@hom**e a částečně i **Sheila**. **Sheila** však pracuje výhradně s emailovou schránkou a celkově se odchyluje od koncepce shadow honeypot (3.6).

Můj výběr tak spočinul na dvou aplikacích: **Argos** a **honey@home**. Analýzou, instalací, použitím a otestováním těchto nástrojů se zabývají následující kapitoly 5 a 6. Testování bude primárně probíhat na 32-bitové linuxové distribuci Arch Linux s verzí jádra 2.6.30. V případě potřeby bude ještě využit server s 32-bitovou linuxovou distribucí Debian Squeeze s verzí jádra 2.6.26. Pokud bude mít aplikace verzi i pro platformu Windows, bude testována na operačním systému Microsoft Windows XP obsahujícím Servis Pack 3 a v té době nejnovější aktualizace.

[7] Na daném portu by tedy měla běžet jak skutečná služba, tak honeypot připravený v případě potřeby zasáhnout, viz 3.6.

Kapitola 5

honey@home

Tato aplikace je klasický honeypot, využívající prázdného adresového prostoru v síti. Myšlenka je taková, že jsou honeypotu v konfiguraci nastaveny určité služby, jež se budou útočníkovi či malwaru jevit jako otevřené. Typické služby, na kterých může honeypot naslouchat, shrnuje tabulka 3.2. honey@home poté data získaná při útoku posílá na centrální server, jehož identifikace je uvedena v konfiguračním souboru.

5.1 Instalace

K dispozici jsou dva instalační balíčky. Jeden pro Linux, druhý pro Windows, proto zde bude popsán zvlášť postup pod jedním operačním systémem a zvlášť pod druhým. To z důvodu, že postup zprovoznění se pod každým operačním systémem dosti liší a s každou verzí jsem řešil různé problémy.

5.1.1 Linux

Na oficiálních stránkách[1] je v sekci Downloads na stažení *.tgz* archív s programem. Ten obsahuje již kompilovanou aplikaci v binárním formátu ELF[2] pro 32-bitovou architekturu. Z toho vyplývá první závažnější nedostatek této implementace a to uzavřenost kódu. Zdrojové kódy aplikace nejsou na jejích oficiálních stránkách k dispozici. Konfigurační soubor aplikace je v textovém formátu[3].

V souboru *README* je uveden krátký postup instalace, kde je mimo jiné zmíněno, že aplikace vyžaduje 32-bitový Linuxový systém s jádrem 2.6.X. Soubor *VERSION* obsahuje verzi programu. Linuxová verze je označená jako *BETA BUILD 2009032314*, tedy beta verze z data 23.3.2009.

Pro použití aplikace je nutné se zaregistrovat na jejích oficiálních stránkách[4]. Registraci je nutné provést na validní emailovou adresu s uvedením jména společnosti a názvu státu. Po registraci uživatel obdrží soubor *user.key*, který v textové formě obsahuje 32-znakový klíč. Soubor s klíčem je poté nutné umístit do adresáře s aplikací. Nyní by měl být nástroj **honey@home** připravený k použití.

[1] http://www.honeyathome.org
[2] Executable and Linkable Format - formát spustitelných souborů používaný v systému Linux.
[3] Jak bývá v unixových systémech zvykem.
[4] http://www.honeyathome.org/test/captcha.php

5.1.2 Windows

Na oficiální stránce projektu je ke stažení také instalátor pro operační systém Microsoft Windows. Aplikace pod tímto operačním systémem vyžaduje *.NET framework 2.0*, jelikož je jejím implementačním jazykem *C#*. Dále je nutné mít nainstalovanou knihovnu *WinPcap*, která aplikacím umožňuje zachytávat pakety na síťovém rozhraní. Instalátor je proveden formou *.msi* souboru, jehož instalace je triviální.

5.2 Použití

Instalace pod oběma operačními systémy se zdařila. Dosud vše probíhalo v pořádku, nyní se však začaly objevovat komplikace.

5.2.1 Linux

Před vlastním spuštěním aplikace se nejprve zaměřím na popis konfiguračního souboru *honeyathome.config*. Ten obsahuje především IP adresu centrálního serveru, na který se data odesílají. S tímto souvisí další nevýhoda této implementace. Centralizované zpracování může být výhodou, avšak v našem případě tak ztrácíme nad vlastními daty kontrolu, jelikož jsou zpracovávána na cizím serveru.

Dále je v konfiguračním souboru možné nastavit honeypotu IP adresu v rámci naší sítě. Honeypot si IP adresu může nechat přidělit dynamicky od DHCP serveru, což je implicitní volba. Nechybí zde uvedení, na jakém síťovém zařízení bude honeypot komunikovat. Tato volba by se měla shodovat se síťovým zařízením, přes jaké jsme do dané sítě, kde chceme honeypot používat, připojeni.

V konfiguraci je také možné nastavit si připojení přes síť TOR[5]. Tato volba je však v implicitním nastavení vypnuta. Dále je v konfiguračním souboru možné nastavit MAC adresou, pod kterou bude honeypot vystupovat. Poslední a nejdůležitější je volba *portmon*, která obsahuje seznam čísel portů, na kterých bude honeypot naslouchat. Implicitně je nastavený pouze port 80.

Nyní se dostáváme k samotnému spuštění aplikace. Aplikace musí být spuštěna s právy uživatele root, jelikož čte data ze síťového rozhraní. Po spuštění aplikace vypisuje ladící (debug) výpisy, z nichž je patrná činnost po spuštění. Ihned po spuštění je kontrolována aktuálnost použité verze[6]. V případě, že je k dispozici novější verze, je aplikace ihned updatována.

Poté aplikace ověří připojení k centrálnímu serveru a provede autentizaci pomocí klíče získaného při registraci. Toto se děje přes zabezpečené SSL spojení. Nakonec se honeypot přes nastavené rozhraní připojí s danou IP a MAC adresou. Na výpisu 5.3 je vidět chování aplikace při spouštění.

Výpis 5.1: honey@home: Výpis při spuštění aplikace

```
$ sudo ./home
...
[Wed Feb 17 12:20:28 2010] wait_for_update():
                           Checking for Update NOW!
```

[5] http://www.torproject.org/

[6] Nejnovější verzi si aplikace zjistí dotazem na http://www.honeyathome.org/downloads/updrepos/current_version.html

```
[Wed Feb 17 12:20:28 2010] check_for_update_():
                           Done checking Versions!
                           Updates_Exist: 0
...
[Wed Feb 17 13:01:43 2010] parseConfig(): Static Address \
                           set to: 192.168.0.33
[Wed Feb 17 13:01:43 2010] parseConfig(): Static MAC Address \
                           set to: '00-11-22-33-44-55'
[Wed Feb 17 12:20:28 2010] parseConfig(): Monitoring Port: 21
[Wed Feb 17 12:20:28 2010] parseConfig(): Monitoring Port: 22
[Wed Feb 17 12:20:28 2010] parseConfig(): Monitoring Port: 80
[Wed Feb 17 12:20:28 2010] parseConfig(): Monitoring Port: 445
...
[Wed Feb 17 12:20:38 2010] OpenConnection() SSL:
                           Host is '139.91.130.199:80'
[Wed Feb 17 12:20:39 2010] client_verifyUser():
         Reading key from file 'XXXXXXXXXX...'. Size of key: 32
[Wed Feb 17 12:20:39 2010] client_verifyUser():
                           Sending key to server...
[Wed Feb 17 12:20:39 2010] client_verifyUser():
                           Received Message: 'YES!'
[Wed Feb 17 12:20:39 2010] client_verifyUser():
                           User Authenticated!
...
DONE!
```

Z výpisu je tedy vidět, že honeypot začal naslouchat na staticky zadané IP adrese **192.168.0.33** s MAC adresou **00:11:22:33:44:55** a naslouchání na portech **21** (FTP), **22** (SSH), **80** (HTTP) a **445** (SMB). Očekávání jsou tedy následující. Na adrese 192.168.0.33 by se nyní měl jevit běžící stroj s otevřenými porty **21, 22, 80** a **445**. Výpis síťového scanneru portů (5.2) však ukazuje něco trochu jiného.

Výpis 5.2: honey@home: Ověřování stavu portů honeypotu.

```
$ ./portScanner -p 21,22,80,445 192.168.0.33

Ports on 192.168.0.33:
PORT    STATE    SERVICE
21/tcp  filtered ftp
22/tcp  filtered ssh
80/tcp  filtered http
445/tcp filtered microsoft-ds
MAC Address: 00:11:22:33:44:55
```

Stav portů je *filtered*, což podle oficiální dokumentace nástroje značí, že program nebyl schopný určit, zda je port otevřený či ne. Nejčastěji se tak děje kvůli zásahu firewallu. Na stroji, z něhož bylo scanování spuštěno, běží linuxový firewall *iptables*, který však v implicitní konfiguraci nemá žádná blokující pravidla. Na routeru, který testovací síť 192.168.0.0/24 vytváří, je firewall také deaktivován. Na žádný z definovaných portů se nelze připojit ani přes telnet či netcat. Po stisku klávesy ENTER v okně konzole s běžícím honeypotem se vypíší statistiky přenosu paketů v tomto tvaru:

Výpis 5.3: honey@home (Linux): Výpis při spuštění aplikace

```
[Wed Feb 17 14:20:15 2010] Linux Honey@Home is up & running!
* IP: '192.168.0.33'
* Status: '7:Ready'
* Online Since: 'Wed␣Feb␣17␣14:07:16␣2010'
* Last Update: 'Wed␣Feb␣17␣14:16:25␣2010'
* Recovery Incidents: 0
* Packets:TCP(0), UDP(0), IP(0), ICMP(0),UNKNOWN(0),INVALID(0)
* Bytes  :TCP(0), UDP(0), IP(0), ICMP(0),UNKNOWN(0),INVALID(0)
```

Po všech možných pokusech o zaslání dat na dané porty ukazují tyto výpisy stále nulový počet paketů všech typů. Aplikace byla poté testována také na Linuxové distribuci **Debian** s verzí jádra **2.6.26** a to jak na systému běžícím na nativním hardwaru, tak i na virtualizové verzi. Pokaždé však se stejným výsledkem.

5.2.2 Windows

Na rozdíl od linuxové verze disponuje verze pro Windows grafickým uživatelským rozhraním napsaným v C# .NET. Také k provozu této verze je nutné do jejího adresáře zkopírovat soubor obsahující licenční klíč. Dalším rozdílem oproti linuxové verzi je implicitně nastavené připojení se přes síť TOR. Jak je však vidět z výpisu 5.4, aplikace má s tímto nemalé problémy a každý pokus o připojení je po čase odmítnut.

Výpis 5.4: honey@home (Windows): Výpis při spuštění aplikace

```
[2010/01/25 10:31:50] Starting H@H engine
...
[2010/01/25 10:31:55] Proxy mode enabled - launching TOR
[2010/01/25 10:31:55] Checking port 9051
[2010/01/25 10:31:56] TOR launched on port 9051
[2010/01/25 10:32:01] Starting TOR Watch thread
[2010/01/25 10:32:01] connecting to NoAH server
[2010/01/25 10:34:03] TOR Request Rejected or Failed Exception
[2010/01/25 10:34:13] connecting to NoAH server
[2010/01/25 10:36:13] TOR Request Rejected or Failed Exception
[2010/01/25 10:36:23] connecting to NoAH server
...
```

Řešení spočívá v nastavení přímého připojení na centrální server, stejně jako to dělá linuxová verze. Verze pro Windows se již nekonfiguruje přes textové konfigurační soubory, ale přes grafické uživatelské rozhraní. Po zrušení volby nastavující připojování se přes síť TOR a nastavení přímého připojení na centrální server následuje potvrzení této akce. To je však vždy provázeno neošetřenou chybou (Unhandled Exception) aplikace a volba se tak nikdy nepotvrdí. Toto chování se projevovalo na obou testovacích strojích s Windows XP SP3 a Windows Vista Business SP1 oba s .NET Framework 3.5. Na Windows XP byl zkušebně proveden downgrade .NET balíčku na doporučenou verzi 2.0, avšak stále se stejným výsledkem.

5.3 Shrnutí

Aplikaci **honey@home** se nepodařilo plně zprovoznit na žádném ze dvou podporovaných operačních systémů. Instalace byla pod oběma systémy bezproblémová, avšak při použití aplikace se začaly objevovat komplikace. Pod Linuxem se aplikace prostřednictvím výpisů informací při běhu tváří, že funguje korektně. Žádané porty však nejsou otevřené a honeypot tak neregistruje žádná data. Pod Windows jsou problémy již se samotnou konfigurací aplikace, kdy implicitně nastavené a nefunkční připojení přes TOR nelze změnit na přímé připojení se k centrálnímu serveru.

I pokud by se nástroj **honey@home** podařilo plně zprovoznit, není přesto pro náš účel příliš vhodný, jelikož nesplňuje několik důležitých požadavků. K aplikaci není dodáván zdrojový kód a tak není možné si chování aplikace v případě potřeby více uzpůsobit. Dalším problémem je centralizovaná správa dat, která je pod kontrolou cizí organizace. Poslední překážkou je, že tato aplikace nesplňuje charakteristiky shadow honeypotu. Větší naděje z tohoto ohledu jsou vkládány do nástroje Argos, kterému se věnuje následující kapitola.

Kapitola 6

Argos

Nástroj Argos je produktem týmu expertů z nizozemské *Faculty of Sciences, VU University Amsterdam*. Tento tým vede PhD. student Georgios Portokalidis. Podle oficiálních stránek bylo cílem vytvořit nástroj pro zachycení tzv. zero-day[1] útoků. Teoretické základy aplikace vycházejí v poznatků projektu NoAH. Samotná implementace započala začátkem roku 2007 a poslední update proběhl v dubnu roku 2009, tedy více než rok po oficiálním ukončení projektu *NoAH*. Poslední příspěvek v argos-devel mailing listu[2] pochází z prosince roku 2009, z čehož předpokládám, že vývoj aplikace již ustal. Přesto jsem nikde nenašel žádnou oficiální zmínku o ukončení tohoto projektu. Poslední verze nástroje Argos nese označení **Argos 0.4.2**.

Argos je z velké části založen na emulačním softwaru **Qemu**, který bude stručně popsán v podkapitole 6.6. Argos byl původně založen na verzi **Qemu 0.8**, ve verzi 0.4 byl však portován na tehdy aktuální **Qemu 0.9**. Pro úplnost, v současnosti je nejnovější verzí[3] **Qemu 0.12**.

Aplikace tedy dokáže emulovat operační systém se všemi jeho službami. Navíc poskytuje vrstvu, která analyzuje síťová data a sleduje jejich použití v operační paměti. Právě tato vrstva, které se budu věnovat v podkapitole 6.3.1, je zodpovědná za včasné rozpoznání útoku. Po zaregistrování útoku Argos vygeneruje log soubor obsahující otisk části paměti figurující v útoku, obsah registrů procesoru a další užitečné informace pro pozdější forenzní analýzu. Strukturou generovaných log souborů se bude věnovat podkapitola 6.5.

Po představení aplikace z teoretické stránky bude následovat popis instalace a použití nástroje v podkapitole 6.8. V samotném závěru kapitoly bude na Argos v laboratorních podmínkách veden útok a analyzována jeho reakce (6.9).

6.1 Cíle nástroje

Argos byl navržen jako nástroj pro automatizované zachytávání nových útoků (tzv. zero-day attacks) a generování jejich signatur. Nástroje s tímto účelem již existují, avšak stále trpí jedním základním nedostatkem a to velkým množstvím tzv. *false positive* hlášení, tedy zachycením neexistujících útoků (viz tabulka 3.1).

Argos se nezaměřuje na útoky způsobené zneužitím špatně nakonfigurovaných služeb, ale na automatizované síťové útoky zneužívající některé ze zranitelností kódu. Argos se

[1] Útok na zranitelnost, která dosud nebyla popsána.

[2] Jedná se o využití emailu k šíření informací mezi mnoho účastníků diskuse. Často se používá právě při vývoji softwaru k poskytnutí zpětné vazby.

[3] K 18.únoru 2010

také nezaměřuje na *payload*[4], ale na vlastní exploit. Jinými slovy, Argos je schopný zachytit vlastní kód zneužívající zranitelnost, ale už neřeší, co by se dělo dále po úspěšně provedeném útoku. Autoři tento krok zdůvodňují tím, že bez vlastního exploitu by útok ani nebyl možný. Navíc stejný exploit může obsahovat různé *payload* kódy, které jde navíc ještě zpolymorfizovat lépe než kód exploitu.

Autoři nástroje Argos si tedy dali za cíl vytvořit nástroj splňující tyto požadavky [10]:

1. Spolehlivá detekce širokého spektra nových útoků
2. Spolehlivé generování signatur, které mohou být poté použity pro zastavení daného typu útoku
3. Finančně nenáročné nasazení do provozu

Vysoké množství *false positives* porušuje první dva body. Tento postup může být s úspěchem použit u IDS systémů (viz kapitola 3.1) avšak pro plně automatizovaný nástroj je nevhodný. Právě přístup označovaný jako ***Dynamic taint analysis*** je takový, který se snaží zcela odstranit či alespoň radikálně minimalizovat počet *false positives*. Tento přístup bude rozebrán v podkapitole 6.3.1.

Základní vlastnosti nástroje jsou následující [10]:

- Nabízí softwarovou ochranu celého operačního systému
- Podporuje komplexní paměťové operace jako memory mapping a DMA[5] zvládá tak odhalovat i velice sofistikované metody útoků.
- Útoky typu *buffer overflow* a *string/code injection* automaticky spouští generování signatur útoku, které mimo jiné obsahují informace odkud a jaká data ze sítě přišla.
- Jakmile je zaregistrován útok, Argos do napadeného procesu injektuje vlastní forenzní shellcode, který dále pomáhá zjistit dodatečné informace o útoku a je použit při generování signatur.

6.2 Příbuzné projekty

Přístup *Dynamic taint analysis* je z velké části kombinací dvou již existujících postupů. Jsou jimi metody/nástroje zvané *Minos*[4] a *Vigilante*[3].

6.2.1 Minos

Minos negeneruje žádné signatury útoků a navíc pro její efektivní nasazení je nutné spoléhat na její implementaci přímo v hardwaru. Metoda pracuje pouze s fyzickými adresami a nemá prostředek pro přímé převody mezi fyzickými a virtuálními adresami. Přestože je schopná zachytit několik typů útoků, má problém s určením zdroje útoku, což je z hlediska automatizovaného zpracování útoků velký handicap.

[4] Útočníkův kód vykonaný po úspěšné exploitaci.
[5] Direct Memory Access: Způsob přístupu k datům bez účasti procesoru.

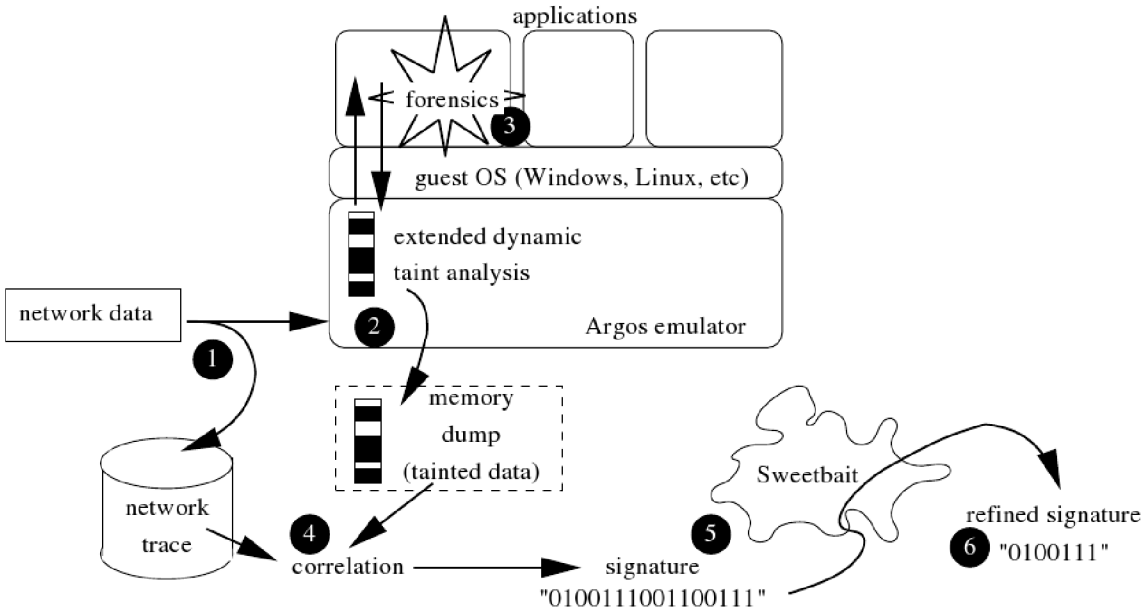
6.2.2 Vigilante

Druhou metodou, která byla nástroji Argos inspirací je metoda Vigilante. Ta pracuje výhradně s virtuálními adresami, což ji limituje například tím, že nepodporuje DMA[6]. Vigilante se zaměřuje na jednotlivé služby (procesy) a vůbec nechrání jádro operačního systému. Na rozdíl od předchozí tato metoda podporuje vytváření signatur útoků.

Autoři nástroje Argos tak kombinují hardwarově orientované řešení na ochranu systému jako celku zvané **Minos** a softwarové řešení zaměřené na jednotlivé procesy zvané **Vigilante** s cílem dosáhnout splnění tří bodů zmíněných na začátku kapitoly a vzít si z obou metod to nejlepší.

6.3 Činnost nástroje

Samotný princip činnosti poměrně výstižně znázorňuje obrázek 6.1. Argos se zaměřuje na monitorování dat proudících směrem ze sítě a jejich následným zneužitím při útoku. Tok síťových dat je předáván komunikující aplikaci a zároveň ukládán ve sledovací databázi, odkud jsou v případě útoku extrahována cenná data. Tento fakt je na obrázku znázorněn číslem **1**. Jakmile se síťová data začnou kopírovat do operační paměti, je použita metoda *Dynamic Taint Analysis* (viz 6.3.1) pro odhalení útoku (na obrázku pod číslem **2**). V případě, že je odhalen útok, jsou postupně provedeny kroky **3-6**, které útok podrobně zaznamenají (viz 6.3.2) do souboru.



Obrázek 6.1: Znázornění činnosti nástroje Argos. Převzato z [10].

[6]Direct Memory Access

6.3.1 Dynamic Taint Analysis

Tato metoda odhalování útoků byla poprvé publikována v [7]. Autoři nástroje Argos používají její modifikovanou verzi, kterou označují jako *Extended Dynamic Taint Analysis*[7]. Princip této metody spočívá v označování dat jako tzv. poskvrněná[8] (*tainted*). Výraz *tainted* značí to, že data jsou nedůvěryhodná a mohou být součástí potenciálního útoku. Jako *tainted* jsou v rámci této metody označena všechna data přicházející ze sítě. Tato data jsou poté důkladně sledována. Pokud dojde ke zkopírovaní takových dat například do paměti či registru procesoru, jsou tyto lokace označeny taktéž jako *tainted*. Pokud dojde k využití takto označených dat například jako místo skoku v rámci jazyka symbolických instrukcí, či jejich jinému nelegálnímu použití, je vyvolán alarm, který následně spustí rutinu zaznamenávající útok do souboru.

Naprostá většina útoků spočívá v přesměrování toku programu na kód vložený útočníkem. Z toho vyplývá, že největší pozornost je věnována registru **EIP**, který ukazuje na následující vykonávanou instrukci. Adresa se do registru **EIP** nahrává instrukcemi *CALL*, *RET* a *JMP*. Monitorování těchto instrukcí v souvislosti s *tainted* daty je tak jedním z hlavním principů metody *Dynamic Taint Analysis*.

Přestože toto opatření zachytí velkou škálu typů exploitů, stále ponechává nepovšimnutou jednu zásadní kategorii útoků. Jedná se o zranitelnosti formátovacího řetězce, kdy útočník může svými daty přepsat téměř libovolnou paměťovou lokaci. V tomto případě se však nejedná o data ze sítě a výše zmíněný princip je tak neúčinný.

Z tohoto důvodu autoři nástroje Argos rozšířili metodu *Dynamic Taint Analysis* o schopnost rozpoznat injektování kódu a zaregistrovat tak i výše zmíněný typ útoků.

Posledním typem útoků rozpoznatelným nástrojem Argos jsou útoky založené na podvrhnutí parametru funkce systémových volání. To je jednoduše realizováno označením parametrů kritických funkcí jako *tainted*. Jedná se hlavně o funkce z rodiny **exec*()** jako jsou **execl()**, **execv()**, **execle()**, **execve()** a další.

Při implementaci této metody byli autoři postaveni před otázku, jak velké budou nejmenší bloky dat označitelné jako *tainted*. Čím menší takové bloky budou, tím bude metoda přesnější, avšak zároveň ji bude zpomalovat větší režie. Co se týče paměťových bloků, rychlost metody byla pro bloky o velikost 1,2 a 4 bajty téměř shodná. Proto byla zvolena hranice jeden bajt. V případě registrů procesoru se však rychlost metody se zmenšujícím se označitelným blokem rapidně snižovala. Z toho důvodu autoři zvolili pouze dva stavy registru a to že nebude označen vůbec nebo bude označen jako *tainted* celý[10].

6.3.2 Zaznamenání útoku do souboru

Jakmile metoda *Dynamic Taint Analysis* odhalí útok, nepovolí zneužití síťových dat a útok tudíž selže. Navíc je do virtuálního adresového prostoru postiženého procesu injektován *shellcode* (vzhledem k výše zmíněnému obrázku je to krok **3**). Ten je závislý na typu emulovaného operačního systému, což je důvod proč je tato informace předávána nástroji Argos jako jeden z povinných parametrů. *Shellcode* tak může pomoci odhalit další informace o útoku, jakými jsou například název napadeného procesu, jeho pid[9], soubory a sockety otevřené daným procesem a další užitečné informace. Podle [10] je to vůbec první využití *shellcode* pro defenzivní účely.

[7] V následujícím textu bude pojem *Dynamic Taint Analysis* označovat metodu použitou v nástroji Argos.
[8] Dále v textu budu používat pouze originální označení *tainted*.
[9] Proccess identification number

Po injektování *shellcode* jsou uložena data poskytující informace o paměťových blocích a registrech procesoru, které byly použity při útoku. V případě aktivního *Network Tracker* módu (viz 6.4) jsou k těmto informacím navíc přidána data přímo ze sítě a je tak možné vyhledat konkrétní síťový paket, který škodlivá data na emulovaný počítač zanesl (krok **4**).

Důležité je také zmínit, že soubory generované nástrojem **Argos** jsou binární. Výhodou tohoto řešení je menší velikost generovaných souborů, avšak za cenu ztíženého zpracování informací z těchto souborů.

6.4 Network Tracker mód

Jak již bylo řečeno v předchozí sekci, **Argos** po zaznamenání útoku vygeneruje soubor se záznamem útoku, který obsahuje především hodnoty základních registrů procesoru a bloky paměti, které sehrály svoji roli při útoku. **Argos** jde však dále a podporuje tzv. *Network Tracker* mód[10].

V tomto módu **Argos** navíc zaznamenává data přicházející ze sítě a jejich spojitost s útokem. Při zapnutém *Network Tracker* módu je tak vygenerovaný soubor obohacen o velmi cenné informace. Argos v průběhu útoku zachycuje síťovou komunikaci do souboru *argos.netlog*. Do souboru *argos.csi.random*, který se váže ke konkrétnímu útoku, pak navíc ke každému registru uloží index, který umožňuje v souboru *argos.netlog* vyhledat konkrétní síťový paket, ze kterého se daná hodnota do registru nakopírovala.

Soubor *argos.netlog* je společný pro všechny log soubory zaznamenaných útoků.

6.4.1 Aktivace Network Tracker módu

Přestože je v oficiální dokumentaci k nástroji **Argos** *Network Tracker* mód zmíněn[11], nikde není uvedeno, jak tento mód zprovoznit. Původně jsem hledal odpovídající parametr, se kterým by se nástroj v tomto módu spouštěl, avšak bez úspěchu. Odpověď jsem nakonec našel na Argos mailing listu[12].

Použití *Network Tracker* módu se neaktivuje parametrem programu, ale zadává se jako parametr skriptu *./configure* použitého při kompilaci. Jak přeložit program s aktivovaným *Network Tracker* módem tak ukazuje výpis 6.1.

Výpis 6.1: Argos: Aktivace NetTracker módu při kompilaci

```
./configure --enable-net-tracker
```

Použití kompilačního parametru místo parametru programu je zdůvodněno tím, že tento mód má velký vliv na výkon programu. **Argos** při zapnutém *Network Tracker* módu spotřebuje přibližně dvakrát tolik paměti. Další nevýhodou je omezení velikosti operační paměti emulovaného stroje na **512 MB**. Tyto nevýhody jsou však nicotné v porovnání s důležitostí informací, jaké tento mód o provedeném útoku poskytne.

[10] Zkráceně označovaný jako *NetTracker* mód

[11] Především v sekci o interpretaci log souborů.

[12] http://mailman.few.vu.nl/pipermail/argos-devel/2006/000043.html

6.5 Interpretace log souborů

Nástroj **Argos** vždy po úspěšném zaregistrování útoku vytvoří v adresáři, odkud je spouštěn, soubor *argos.csi.random*, kde random je náhodné celé číslo. Tyto soubory obsahují tzv. *memory fingerprint* (otisk paměti) detekovaného útoku. Je do nich exportován veškerý stav systému, který byl útokem ovlivněn. **Argos** zjistí, který proces byl napaden a vyhledá pouze stránky paměti, které jsou dostupné tomuto procesu. Tím redukuje objem dat a vylučuje data, která s útokem nesouvisejí.

Pokud je nástroji pomocí parametru sděleno, jaký operační systém v emulovaném stroji běží (což je doporučeno), je schopen rozlišit mezi uživatelskou částí paměti (*user memory space*) a pamětí jádra systému (*kernel memory space*) a díky tomu dále vyloučit nesouvisející data a tím redukovat objem exportovaných souborů. Tabulky v následující podkapitole se strukturou jednotlivých částí log souboru jsou převzaty z dokumentace nástroje **Argos**[13].

6.5.1 Struktura log souboru

Všechny log soubory mají pevně danou strukturu, která začíná *hlavičkou logu*, za níž se vyskytují vždy dvojice *hlavička paměťového bloku* a *obsah paměťového bloku*. Soubor je ukončen první prázdnou hlavičkou paměťového bloku. Detailní popis poskytuje tabulka 6.1.

LOG HEADER	
MEMORY BLOCK HEADER	MEMORY BLOCK CONTENTS
...	
EMPTY MEMORY BLOCK HEADER	

Tabulka 6.1: Uspořádání log souboru generovaného nástrojem Argos

6.5.2 Struktura hlavičky log souboru

Nyní bude podrobně popsána pouze část *LOG HEADER*, tedy hlavička log souboru. Hlavička začíná *číslem verze*, které ovlivňuje formátování zbytku hlavičky a souboru. To je z toho důvodu, že formátování log souborů se v jednotlivých verzích Argosu měnilo. Za číslem verze následuje typ architektury emulováno systému, jež byl obětí útoku. Toto pole také ovlivňuje další uspořádání hlavičky a zbytku log souboru. Další dvě pole udávají typ útoku a časové razítko, kdy k útoku došlo. Typy útoků, které dokáže Argos zaznamenat, jsou uvedeny v tabulce 6.2.

Následující část hlavičky je závislá na architektuře napadeného systému. Jako první jsou uloženy obsahy základních registrů procesoru. Na **X86** architektuře to je osm základních (*general purpose*) 32-bitových registrů, zatímco na **X86_64** to je šestnáct základních 64-bitových registrů. Registry jsou uloženy v tomto pořadí: **EAX, ECX, EDX, EBX, ESP, EBP, ESI** a **EDI**.

[13] http://www.few.vu.nl/argos/docs/logs.html

ALERT_JMP	Podstrčená data byla použita v JMP instrukci.
ALERT_LJMP	Podstrčená data byla použita v JMP instrukci v chráněném módu (zastaralé)
ALERT_TSS	Podstrčená data byla nahrána do EIP registru.
ALERT_LCALL	Podstrčená data byla použita v CALL instrukci reálného módu.
ALERT_IRET	Podstrčená data byla použita v IRET instrukci.
ALERT_RET	Podstrčená data byla použita v RET instrukci.
ALERT_WRMSR	Podstrčená data byla zapsána do MSR registru (zastaralé)
ALERT_CI	Tok programu byl přesměrován na instrukci ovlivněnou podstrčenými daty.

Tabulka 6.2: Typy útoků zaznamenatelné nástrojem Argos.

Pole s obsahem registrů je následováno polem udávajícím vlastnosti jednotlivých registrů. Každá taková vlastnost udává, zda byl obsah registru součástí útoku a pokud byl, tak odkud z fyzické paměti byl nahrán. Položka vlastností má pro každý registr stejnou velikost. Nenulová položka značí, že registr byl součástí útoku a hodnota této položky udává adresu paměti, odkud byla hodnota do registru nahrána. Hodnota -1 (*0xffffffff*) značí, že hodnota byla načtena rovnou ze sítě.

Za obecnými registry procesoru v hlavičce následuje obsah registru **EIP**, spolu s jeho vlastnostmi. Ve většině případů je v této vlastnosti zaznačeno, že **EIP** měl podíl na útoku a obsahuje adresu podstrčenou útočníkem. To je obvykle adresa paměti, kam došlo k injektování shellcode[14].

Posledním polem v hlavičce log souboru je pak obsah status registru **EFLAGS**, jehož velikost také závisí na architektuře (32 nebo 64 bitů).

Přehledný popis struktury části *LOG HEADER* poskytuje tabulka 6.3. V této tabulce jsou navíc vidět pole *NET TRACKER VALUES, NET TRACKER VALUES* a *EIP NET TRACKER TAG*, která nebyla výš popsána. Přítomnost těchto polí je indikována nastavením osmého bitu v poli *FORMAT*. Tato pole slouží k vysledování, odkud data po síti přišla a v log souborech jsou přítomné pouze pokud je Argos spuštěn v tzv. Network Tracker módu, o kterém pojednávala sekce 6.4.

6.5.3 Struktura hlavičky bloku paměti

Tato část se bude zabývat strukturou hlavičky bloku paměti, v tabulce 6.1 označené jako *MEMORY BLOCK HEADER*. Stejně jako hlavička celého log souboru tak i hlavička bloku paměti začíná číslem verze, které následně ovlivňuje další strukturu dat.

Dalším polem je opět indikátor toho, zda se tato část paměti podílela na útoku. Ve většině případů, by tento ukazatel měl značit, že daný paměťový blok byl využit při útoku. To je ostatně cílem log souborů, protože paměťové bloky nesouvisející s útokem pro nás

[14]Shellcode je kód, který se vykoná po úspěšném zneužití zranitelnosti.

FORMAT	ARCH	TYPE	TIMESTAMP
REGISTER VALUES			
TAGS VALUES			
(NET TRACKER VALUES)			
EIP VALUE		EIP TAG	
(EIP NET TRACKER TAG)			
EFLAGS VALUE			

Tabulka 6.3: Uspořádání hlavičky log souboru generovaného nástrojem Argos.

nemají takový význam. Přesto může nastat případ, že EIP registr ukazuje do místa v paměti, které není označeno jako podílející se na útoku. Takové paměťové místo je pak do log souboru uloženo také s tím, že značka indikující využití při útoku je nastavena na nulu.

Třetí pole udává velikost dat, která následují za hlavičkou bloku paměti. Dále následují pole udávající adresu paměťového bloku v operační paměti. Zde je zaznamenána jak fyzická tak virtuální adresa bloku. Struktura hlavičky bloku paměti je popsána v tabulce 6.4

FORMAT	TAINTED	SIZE
PADDR		VADDR

Tabulka 6.4: Uspořádání hlavičky bloku paměti log souboru generovaného nástrojem Argos.

6.5.4 Struktura dat z Network Tracker

Pokud je **Argos** spouštěn se zapnutým *Network Tracker* módem (viz 6.4), za sekcí bloků paměti v log souboru následuje další sekce týkající se síťových dat a dále je generován společný soubor *argos.netlog*. Strukturu dat v tomto souboru popisuje tabulka 6.5.

Popis	**Velikost**
ETHERNET FRAME SIZE	2 bajty (Little endian)
ETHERNET FRAME DATA	Specifikována hodnotou předchozího pole

Tabulka 6.5: Struktura dat v log souboru *argos.netlog*.

6.5.5 Shrnutí

Nástroj **Argos** generuje binární soubory obsahující cenné informace o provedeném útoku. Použití binárního formátu namísto textového snižuje velikost generovaných souborů, avšak mírně komplikuje zpracování dat z těchto souborů. V této sekci byla detailně popsána struktura těchto dat. Díky těmto znalostem je tak mnohem snazší požadované informace z log souborů dostat. Popis aplikace, která tato data zpracovává, bude uveden v praktické části této práce (kapitola 7).

Ze záznamu jednoho incidentu je možné získat velké množství dat. Pro další využití z hlediska automatizovaného zpracování útoků jsou pak některá data důležitější než jiná. Z informací o registrech jsou zdaleka nejdůležitější data týkající se registru **EIP**. Registr **EIP** zvaný *Instruction Pointer* ukazuje na další vykonávanou instrukci z hlediska toku programu. Právě hodnota tohoto registru bývá útočníkem nejčastěji podstrčena a tím je docíleno přesměrování toku programu na kód vložený útočníkem. Falešná hodnota registru *EIP* se do registru dostane z paměti RAM. Tam se zase nejčastěji v případě síťového útoku dostane zkopírováním ze síťového paketu/paketů poslaných útočníkem. Záznamové soubory útoku umožňují tuto cestu dat zpětně zmonitorovat a zjistit tak, z jakého konkrétního síťového paketu se hodnota nakopírovala do konkrétního paměťového bloku. Právě tyto informace jsou velmi důležité z hlediska pozdějšího zpracování dat.

6.6 QEMU

Jelikož Argos je z velké části založen na unixovém nástroji **QEMU**, bude v této podkapitole stručně představen. **QEMU** je emulátor procesoru, tedy nástroj, který umí emulovat virtuální stroj běžící na určité z podporovaných architektur. Do tohoto virtuálního stroje je pak možné nainstalovat podporovaný operační systém, který samozřejmě musí být kompatibilní z architekturou virtuálního stroje. Základní **QEMU** knihovna pro emulaci CPU pod licencí *GNU LGPL* [15]. Další části jsou pak dostupné pod *BSD* licencí.

Qemu dovoluje kompletně[16] emulovat různé operační systémy. Jmenovitě například Microsoft Windows, Linux, BSD, Solaris, DOS a další. Co se týče emulovaných architektur, podporované jsou například *arm*, *i386*, *m68k*, *mips*, *ppc*, *sparc*, *x86_64* a mnohé další. Kompletní seznam je dohledatelný v oficiální dokumentaci. Samotné Qemu je primárně určeno pro operační systém Linux. Existují však experimentální instalátory i pro Windows a MAC OS X.

QEMU není jedinou aplikací svého druhu. Mezi nejznámější komerční řešení patří **VMware**[17] ve verzích **WMware Workstation** a **WMware Server**. Z open-source pak stojí za zmínku kvalitní **VirtualBox**[18].

6.6.1 QEMU-KVM

Nové verze **QEMU** jsou k dispozici i ve verzi **QEMU-KVM**. Tato verze podporuje pouze architektury i386 a x86_64 a oproti standardní verzi **QEMU** obsahuje KVM[19] modul, který umožňuje emulovat virtuální stroj téměř nativní rychlostí. Standardní **QEMU** trpí velkou ztrátou výkonu virtuálního stroje, proto v případě emulace architektury *i386* nebo *x86_64*

[15] GNU Lesser General Public License
[16] Včetně procesoru a různých periferií
[17] http://www.vmware.com
[18] http://www.virtualbox.org
[19] Kernel Virtual Machine

je velice výhodné použít **QEMU-KVM**. Argos bohužel tento rys nepřevzal, tudíž pro něho platí hlavní nevýhoda klasické verze **QEMU** a to nízký výkon emulovaných systémů. Srovnání rychlostí startu emulovaného 32-bitového stroje s 512 MB RAM pomocí QEMU, QEMU s kernel modulem kqemu a nástroje Argos poskytuje tabulka 6.6. Měření probíhalo od zapnutí virtuálního stroje až do plného načtení OS Windows XP. Test probíhal na počítači s procesorem Intel Core 2 Duo T7300 (2.00GHz) s 3072 MB RAM.

Nástroj	**Čas**
QEMU v0.10.6	1m 19s
QEMU v0.10.6 s kqemu modulem	48s
Argos v0.4.2	4m 23s

Tabulka 6.6: Srovnání rychlosti spouštění virtuální stroje.

6.6.2 Operační módy

QEMU má základní módy, které se liší hloubkou emulace a použitím.

1) User mode emulation

V tomto módu umožňuje **QEMU** spustit Linuxový či MAC OS X program zkompilovaný pro určitý typ procesoru pod jiným typem procesoru. Tento mód slouží hlavně usnadnění činností jako je *cross-compilation* a *cross-debugging*. Tyto dva pojmy se běžně do češtiny nepřekládají. V případě *cross-compilation* se jedná o kompilaci programů na určité architektuře, které jsou však určené pro architekturu jinou. Příkladem může být kompilace aplikace pro architekturu MISP[20] na stroji s klasickým *i386* procesorem. *Cross-debugging* je pak pojem související s laděním takovýchto aplikací.

2) Complete Computer System mode emulation

Tento mód se základním rysem nástroje **QEMU** a umožňuje emulovat celý virtuální stroj včetně periferií, ne jen jednotlivou aplikaci jako předchozí mód. V tomto módu je tedy nutné do virtuálního stroje nainstalovat jeden z podporovaných operačních systémů a až v něm provozovat dané aplikace. Právě tento mód je důležitý v kontextu nástroje Argos, kdy je žádoucí emulovat celý operační systém, který je navíc zapouzdřen do vrstvy rozpoznávající případné útoky.

6.7 Instalace

Jak již bylo řečeno, nástroj **Argos** je open-source a tudíž je šířen ve formě zdrojových kódů. Stejně jako **QEMU**, je i samotná nadstavba **Argos** psána v jazyce C.

[20]MIPS architektura se v současnosti používá například u tiskáren či set-top boxů

6.7.1 Požadavky na systém

Aplikace je podporována pouze pod operačním systémem Linux s verzí **jádra 2.4 nebo 2.6**. Ke kompilaci je třeba mít nainstalovanou **knihovnu SDL**[21], která zajišťuje nízkoúrovňový přístup k 2D a 3D grafice přes OpenGL a dále přístup k zařízením jako například myš, klávesnice či joystick. Samozřejmostí je mít nainstalovaný **překladač GCC**. Oficiální dokumentace se nezmiňuje o doporučené verzi GCC, avšak při pokusu o kompilaci verzí gcc 4.4.1 zobrazí skript **configure** varování, které je zachyceno na výpisu 6.2.

Výpis 6.2: Argos: Kompilace pomocí gcc 4.4

```
$ ./configure
WARNING: "gcc" looks like gcc 4.x
Looking for gcc 3.x
gcc 3.x not found!
ARGOS is known to have problems when compiled with gcc 4.x
It is recommended that you use gcc 3.x to build ARGOS
To use this compiler anyway, configure with --disable-gcc-check
```

Doporučená je tedy verze **gcc 3.X**. K sesíťování virtuálního systému běžícího v **Argosu** s hostitelským počítačem přes *TUN/TAP* je vhodné mít nainstalovanou utilitu **tunctl**. Pro vytvoření disku pro virtuální počítač je doporučené použít nástroj *qemu-img*, který je standardně součástí balíčku s **aplikací QEMU**.

Po nainstalování nutných závislostí, stažení a rozbalení zdrojových kódů se kompilace a instalace aplikace provede klasickou trojkombinací (viz výpis 6.3).

Výpis 6.3: Argos: Kompilace a instalace aplikace

```
./configure && make && make install
```

6.8 Použití

6.8.1 Vytvoření virtuálního počítače

Po nainstalování se aplikace nakopíruje do adresáře */usr/local/bin/*. Prvním krokem ke zprovoznění virtuálního počítače je vytvoření souboru reprezentujícího harddisk virtuálního počítače. Jak již bylo řečeno, k tomuto úkonu je vhodné využít aplikaci **qemu-img** jako to ukazuje výpis 6.4.

Výpis 6.4: Argos: Vytvoření virtuálního HDD

```
$ qemu-img create -f qcow winxp.img 2G
```

Tento příkaz vytvoří soubor *winxp.img* reprezentující 2GB velký harddisk. Nyní je nutné na harddisk nainstalovat operační systém. V našem ukázkovém případě se bude jednat o **Windows XP SP2**. Ačkoliv je možné OS nainstalovat i přes **Argos**, je vhodnější použít

[21] Simple DirectMedia Layer, http://www.libsdl.org/

QEMU z důvodu možnosti použití již zmíněného modulu **kqemu**. Díky tomu proces instalace velice urychlíme. Proces spuštění QEMU s nově vytvořeným virtuálním harddiskem je zachycen na výpisu 6.5.

Výpis 6.5: Argos: Instalace OS

```
qemu -localtime \
     -m 2GB \
     -hda winxp.img \
     -cdrom winxpsp2.iso \
     -boot d
```

Parametr **-m** značí velikost paměti virtuálního počítače, parametr **-hda** udává soubor reprezentující harddisk počítače a parametr **-cdrom** značí cestu k instalačnímu disku s operačním systémem. Zde můžeme zadat buď přímo fyzické zařízení nebo iso obraz disku. Parametr **-localtime** říká virtuálnímu stroji, aby si čas převzal z hostitelského operačního systému. Poslední parametr **-boot** s argumentem **d** udává, že se bude bootovat z disku. Po spuštění virtuálního počítače standardně nainstalujeme operační systém. Zde by mělo vše proběhnout, jako kdybychom systém instalovali na fyzický počítač.

6.8.2 Sesíťování

Jelikož nástroj **Argos** je honeypot, je nanejvýš vhodné ho vystavit veřejně na síti, aby měl možnost sbírat data z útoků. K tomu je nutné **Argos** sesíťovat se sítí hostitelského počítače tak, aby se jevilo, že hostitelský i virtuální počítač jsou dva samostatné počítače v rámci sítě. Samotný hostitelský počítač v rámci sítě není příliš důležitý. Důležité je, že útočník/malware bude mít z naší sítě (případně Internetu) možnost zaútočit na honeypot. **Argos** je sice dodáván se skripty zajišťujícími spojení virtuálního a fyzického stroje, avšak toto spojení je realizováno přes *NAT*[22], což z hlediska vystavení honeypotu není vhodné.

Stručně si tedy popíšeme obecný postup, jak dosáhnout vystavení honeypotu na síti fyzického počítače. Prvním krokem je vytvoření bridge. K tomu je nutné mít nainstalované **bridge-utils**. Vznikne tak síťové rozraní *brX*. Dále je nutné vytvořit *tapX* rozhraní pomocí již zmíněného nástroje **tunctl**. Toto rozhraní poté bude využívat virtuální počítač. Síťové rozhraní hostitelského počítače (většinou *ethX* nebo *wlanX*) a rozhraní *tapX* se poté přidají do bridge *brX*, kde X udává pořadové číslo zařízení. Tím je zajištěno, že se fyzický a virtuální počítač nacházejí ve stejné síti na úrovni ISO/OSI L2 vrstvy. Dále je nutné přepnout síťové rozhraní fyzického počítače do promiskuitního režimu. Je to z toho důvodu, že tato linka je sdílená pro fyzický i virtuální počítač. Posledním krokem je zapnutí funkce *IP forwarding*. Tato vlastnost je totiž v linuxovém jádře standardně vypnutá. *IP forwarding* tak umožní fyzickému počítači chovat se jako gateway a přeposílat pakety, které nejsou určené přímo jemu.

Nastavení výše popsaného zajistí skript z výpisu 6.6. Skript je nutný spouštět s právy uživatele root. V tom ukázkovém případě předpokládejme, že se nacházíme v LAN síti *192.168.0.0/24* s výchozí bránou *192.168.0.1*.

[22] Network address translation, http://en.wikipedia.org/wiki/Network_address_translation

Výpis 6.6: Argos: Nastavení sítě

```
#!/bin/bash

echo 1 > /proc/sys/net/ipv4/ip_forward'

ip␣addr␣flush␣dev␣eth0
brctl␣addbr␣br0
tunctl␣-u␣stoyan␣-t␣tap0
brctl␣addif␣br0␣tap0
brctl␣addif␣br0␣eth0
ip␣li␣set␣tap0␣promisc␣on
ip␣li␣set␣tap0␣up
ip␣li␣set␣eth0␣promisc␣on
ip␣li␣set␣eth0␣up
ip␣li␣set␣br0␣up
ip␣add␣add␣192.168.0.10/24␣dev␣br0
ip␣route␣add␣default␣via␣192.168.0.1
```

Pro úplnost doplním, že ruční přepínání rozhraní *eth0* a *tap0* do promiskuitního režimu není nutné, jelikož se tak stane automaticky po jejich přidání do bridge. Zde je to uvedeno spíše pro názornost. Také nastavení IP adresy bridge není nezbytné, je však vhodné pro další akce týkající se konfigurace bridge.

6.8.3 Spuštění

V tomto okamžiku je operační systém úspěšné nainstalován na virtuální počítač a ten je v jedné síti s fyzickým počítačem. Nyní se tak na řadu konečně dostává **Argos**. Ten přebírá většinu parametrů z nástroje **QEMU**. Spuštění virtuálního stroje přes **Argos** je tak většinou shodné jako jeho spuštění přes **QEMU**. Výjimku tvoří parametry udávající emulovaný operační systém (*-linux*, *-win2k* nebo *-winxp*). Tento parametr je nutný z důvodů korektního logování v případě útoku (viz 6.3.2).

Spuštění virtuálního stroje reflektující výše uvedenou síťovou konfiguraci tak může vypadat tak jako na výpisu 6.7.

Výpis 6.7: Argos: Instalace OS

```
sudo argos -localtime \
           -m 1024 \
           -hda root.img \
           -winxp \
           -net nic,macaddr=00:16:D4:11:22:33 \
           -net tap,ifname=tap0,script=net.sh
```

Argos na rozdíl od **QEMU** je nutné spouštět pod právy uživatele root. První dva parametry byly již popsány u instalace OS do virtuálního stroje. Parametr *-winxp* byl taktéž již zmíněn výše. Parametr *-net nic* vytvoří síťové rozhraní síťové karty virtuálního stroje s MAC adresou 00:16:D4:11:22:33. Parametr *-net tap,ifname=tap0,script=net.sh* pak zařizuje, že virtuální stroj použije námi vytvořené rozhraní *tap0* a také to, že pro konfiguraci sítě se použije skript *net.sh* z výpisu 6.6.

Z virtuálního operačního systému nyní není problém si od DHCP serveru vyžádat IP adresu, pingovat okolní počítače v síti včetně fyzického počítače a přistupovat na Internet.

6.9 Příklad útoku

V této chvíli je již honeypot plně zprovozněný a vystavený na síti. V této kapitole bude tedy demonstrován příklad útoku na operační systém. Útok bude vedený z jiného počítače ve stejném segmentu sítě. Pro vlastní průnik bude využit exploit zneužívající zranitelnost **MS08-67**. Nejdříve tedy pár slov k této zranitelnosti.

6.9.1 MS08-67

O chybě s označením **MS08-67** jsem se zmiňoval již v úvodu této práce. 14. října 2008 vydává Microsoft opravu kritické chyby mimo plánovanou dávku záplat. Ještě dlouhé měsíce poté však zůstává mnoho počítačů neaktualizovaných z čehož těží malware. Tato chyba se řadí do kategorie *buffer overflow*. Česky přetečení zásobníku. Chyba vzniká většinou kvůli chybné kontrole mezí vstupních dat, která se ukládaní do bufferu. Část dat, která přetečou poté přepisují vlastní kód programu. Pokud se útočníkovi povede přepsat i návratovou adresu a nahradit jí jeho vlastní, může spustit jakýkoliv kód s právy zranitelného programu[8].

V případě MS08-67 se chyba se vyskytovala v souboru *netapi32.dll* v proceduře *NetPathCanonicalize()*[23], která sloužila k odstranění přebytečných znaků (tečka, dvě tečky, lomítko a zpětné lomítko) z *URI*[24]. Jinými slovy tak vykonávala proces označovaný jako *path canonicalization*.

6.9.2 Útok na honeypot

K útoku na honeypot použijeme exploit zneužívající zranitelnost MS08-67. Předpokládejme, že honeypot běží na IP adrese *192.168.0.33*. Spuštění exploitu zobrazuje výpis 6.8.

Výpis 6.8: Spuštění exploitu zneužívajícího chybu MS08-67

```
$ ./ms08_067_netapi PAYLOAD=windows/shell/bind_tcp
                    RHOST=192.168.0.33
```

Parametr *PAYLOAD* určuje, jaký kód se pošle a vykoná na zranitelném počítači. Parametr *RHOST* zase udává adresu zranitelného počítače. Nástroj tedy využije zranitelnost na cílovém počítači a vrátí jeho příkazovou řádku.

Nejprve provedeme demonstraci útoku na virtuální stroj spuštěný prostřednictvím **QEMU**. To z důvodu, aby bylo možné porovnat výstup útoku na stroj spuštěný prostřednictvím **QEMU** a prostřednictvím honeypotu **Argos**. Výstup nástroje po útoku na virtuální stroj spuštěný pomocí QEMU je zaznamenaný na výpisu 6.9.

Výpis 6.9: Výstup exploitu po útoku na OS emulovaný pomocí QEMU

```
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:Czech
```

[23] http://cyberwarfaremag.wordpress.com/2008/12/02/new-kid-on-the-block-downadup/

[24] **Uniform Resource Identifier**: Řetězec sloužící k jednoznačné identifikaci zdroje informací.

```
[*] Selected Target: Windows XP SP2 Czech (NX)
[*] Triggering the vulnerability...
[*] Sending stage (240 bytes) to 192.168.0.33

Microsoft Windows XP [Verze 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>
```

Je vidět, že nástroj správně určil verzi operačního systému, využil zranitelnost a zpřístupnil příkazový řádek systému. V případě honeypotu **Argos** se pak výstup exploitu liší, jak je znázorněno na výpisu 6.10.

Výpis 6.10: Výstup exploitu po útoku na OS emulovaný pomocí nástroje Argos

```
[*] Automatically detecting the target...
[*] Fingerprint: Windows XP Service Pack 2 - lang:Czech
[*] Selected Target: Windows XP SP2 Czech (NX)
[*] Triggering the vulnerability...
```

Nevidíme zde otevření spojení ani příkazový řádek. Je tak jasné, že **Argos** zasáhl a exploit selhal. Navíc v okně konzole, odkud byl **Argos** spuštěn, přibyly hlášení zobrazené na výpisu 6.11:

Výpis 6.11: Argos: Hlášení o zaznamenaném útoku

```
[ARGOS] Attack detected, code <RET>PC<77c37eb2>TARGET<6fe216e2>
[ARGOS] Log generated <argos.csi.680888456>
[ARGOS] Injecting forensics shellcode at 0x00082000[0x0aeff000]
```

To jen potvrzuje, že Argos útok zaregistroval a celou událost zaznamenal do souboru *argos.csi.680888456*.

6.10 Shrnutí

Tato kapitola představila nástroj **Argos**, který splňuje téměř všechny podmínky vytyčené v úvodu práce. Nástroj funguje jako shadow honeypot, jsou k němu k dispozici zdrojové kódy, je schopen registrovat širokou škálu útoku a o každém útoku dovede poskytnout množství cenných informací.

Nástroj se podařilo úspěšně nainstalovat a zprovoznit na testovacím stroji. Poté byl honeypot v laboratorních podmínkách podroben řízenému útoku, který proběhl podle očekávání. **Argos** útok zaznamenal a vygeneroval výstupní soubor s informacemi o útoku.

Tento nástroj byl ze všech implementací projektu **NoAH** vybrán jako nejvhodnější a bude použit také pro praktickou část práce, kdy bude rozšířen o možnost automatického zpracování výstupních souborů.

Kapitola 7

Automatizované zpracování útoků

Důležitost automatizace zpracování výstupů útoků byla v této práci již zmíněna. Ruční sběr dat a jejich následné zařazování do databáze je příliš pomalé a v dnešní době nedostatečné. Konceptem, který tato práce sleduje, je naprostá automatizace tohoto procesu. Pod pojmem automatizace zpracování útoků je myšlena tato posloupnost akcí:

1. Malware zaútočí na službu/shadow honeypot s využitím zatím zcela neznámé (*zero day*) zranitelnosti.
2. Shadow honeypot vygeneruje záznam o celé události a tato data jsou následně uložena do databáze.
3. Z nasbíraných dat z útoků jsou poté vytvořena pravidla a signatury, které jsou okamžitě odesílány jako aktualizace do IDS/IPS/ADS systémů či jiných anti-malwarových nástrojů.

Celý tento řetězec událostí pak proběhne v řádu jednotek vteřin a pozdější útok stejného typu malware by tak byl okamžitě rozpoznán a zastaven.

Aplikace typu shadow honeypot analyzovaná v této práci obstojně zapadá do výše uvedeného schématu. Jakmile malware zaútočí na zranitelnost v programu či operačním systému emulovaném v nástroji Argos, ten vygeneruje soubor se záznamem útoku. Argos sám o sobě další automatizaci zpracování útoku již neposkytuje. Zde je tedy prostor pro vytvoření aplikace, která soubory generované honeypotem zpracuje. Toto zpracování spočívá v tom, extrahovat z výstupních souborů data a ta uložit do databáze, odkud budou jednoduše přístupná pro další zpracování.

7.1 Využití dat z útoků

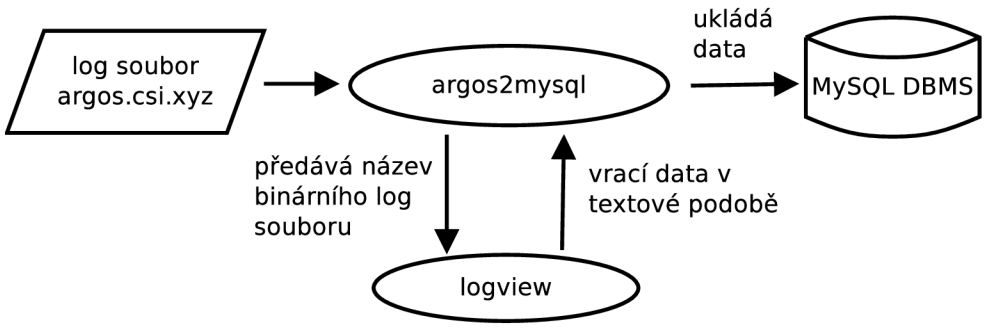
Toto další zpracování spočívá například ve vygenerování signatur jednotlivých útoků. Tyto signatury pak budou v rámci chráněného segmentu sítě distribuovány do systémů IDS/IPS či jiných anti-malwarových nástrojů. Tím by byla zajištěna okamžitá imunizace sítě a jejích koncových zařízení vzhledem k právě zachyceným útokům. Další možností zpracování uložených dat z útoků je využití tzv. dolování dat (*data mining*). Toto téma je však už mimo rozsah této práce a je popsáno například v [5].

7.2 Implementace automatizovaného zpracování útoků

Rozsah implementace programu v rámci této práce pokrývá druhý bod výše zmíněné posloupnosti akcí. Vybraný nástroj Argos generuje binární soubory se záznamem útoku. Nástroj implementovaný v rámci této práce pak provádí extrakci dat z těchto souborů, jejichž formát byl důkladně popsán v kapitole 6.5 a následně se postará o uložení těchto dat do databáze.

Toto řešení jsem rozdělil na dvě spolupracující aplikace. První z nich je nazvaná **logview**, jejímž úkolem je extrakce informací z binární výstupních souborů. Druhá aplikace nazvaná **argos2mysql** pak tato data ukládá do relační databáze.

Spolupráce aplikací při zpracovávání souboru se záznamem útoku je zachycena na obrázku 7.1. Nástroj **argos2mysql** zpracovávající výstupní soubor nástroje Argos předá jméno tohoto souboru aplikaci **logview**. Ta převede informace obsažené v souboru do textové podoby a předá je zpět aplikaci **argos2mysql**, která je následně uloží do databáze.



Obrázek 7.1: Schéma spolupráce aplikací argos2mysql logview

7.3 logview

Aplikace **logview** načítá vstupní data ze specifikovaného Argos log souboru a převede je na textovou reprezentaci, kterou přehledně vytiskne na standardní výstup. Pomocí parametrů lze volit, jaké konkrétní informace z log souboru mají být zobrazeny. Aplikace počítá s tím, že log soubory byly vygenerovány nástrojem Argos se zapnutým *Network Tracker* módem (viz 6.4). Z toho vyplývá, že aplikace hledá ve stejném adresáři, ve kterém se nachází vstupní log soubor, také soubor *argos.netlog*, který obsahuje zaznamenaná síťová data.

7.3.1 Zvolené technologie

Jelikož Argos generuje log soubory v binární podobě, je nutné tyto informace nějakým způsobem převést do textové podoby. K tomu je nutné detailně znát vnitřní strukturu těchto souborů (viz 6.5) anebo použít již hotovou knihovnu poskytující požadované rutiny. V tomto případě se jednalo o druhou ze zmíněných možností. V případě nástroje Argos k němu byla vytvořena také knihovna s názvem **cargos-lib** právě pro tento účel.

Knihovna je napsána v jazyce C a poskytuje pohodlné API[1] pro extrakci požadovaných informací z log souborů. Z toho důvodu je také program **logview**, využívající tuto knihovnu,

[1] Application Programming Interface

psán v jazyce C. Kromě knihovny třetí strany **cargos-lib** byly v aplikaci použity už jen standardní knihovny jazyka C.

7.3.2 Popis aplikace

Na výpisu 7.1 je ukázka nápovědy programu, na které jsou vidět veškeré volby, které **logview** podporuje. Tyto volby budou následně vysvětleny, některé i na praktických ukázkách.

Výpis 7.1: logview: Výpis nápovědy

```
$ ./logview -h
Argos Log Viewer v1.0
Usage:
    logview -f argos.csi.XYZ [-a|-m|-r|-b|-p] [-n] [-h]
    -f Argos log file
    -a Print both header and memory blocks information
    -r Print only information from log header (registers,
       architecture, timestamp,...)
    -m Print memory blocks.
    -b Print memory block where EIP value was loaded from.
    -p Print raw packet where EIP value was loaded from.
    -n Print all whole data lenght. (if not specified, data
       over 20 bytes are truncated)
    -h Print this help.
Note 1: File argos.netlog must be in the same direcotory
        as file specified via -f option!
Note 2: If neither of -a,-m,-r is specified, information
        from log header are printed.
```

Jediným povinným parametrem je parametr **-f** specifikující vstupní soubor se záznamem útoku. Program si pak automaticky ze stejného adresáře načte i soubor *argos.netlog*. Pokud není specifikována žádná jiná volba, *logview* vytiskne pouze základní informace z hlavičky log souboru. Jedná se především o typ útoku, timestamp a použitou architekturu. Co se týče výpisu registrů procesoru, program ke každému registru vypíše jeho hodnotu v momentě útoku, fyzickou adresu paměti, odkud byla tato hodnota nakopírována a příznak signalizující, zda byl registr přímo součástí útoku. V závěru je vypsán obsah stavového registru *EFLAGS* a jako poslední je vypsána hodnota označená jako *FaultyEIP*, která udává registru *EIP* po ovlivnění škodlivými daty. Ve většině případů to tak je ukazatel na první instrukci bloku škodlivého kódu podstrčeného útočníkem. Takový výpis je pak znázorněn na výpisu 7.2.

Výpis 7.2: logview: Výpis základních informací o útoku zneužívajícím nezáplatovanou zranitelnost MS08-67 systému Windos XP SP2

```
$ ./logview -f argos.csi.1009706681
Version:        0x02
Arch:           i386
Type:           RET
Timestamp:      1270295549
Name    Value           MemFromAddr     Tainted
```

```
EAX      0x4954494f      0x07cff45c      1
EBX      0x0104005c      0x07cff49a      1
ECX      0x0104f4a4      0x00000000      0
EDX      0x0104f4fa      0x00000000      0
ESP      0x0104f45c      0x00000000      0
EBP      0x00020408      0x07cff454      1
ESI      0x0104f496      0x00000000      0
EDI      0x0104f444      0x00000000      0
EIP      0x6fe216e2      0x07cff458      1
EFLAGS:    0x00000202
FaultyEIP: 0x77c37eb2
```

Parametr **-m** vytiskne paměťové bloky související s útokem. U každého bloku je vypsána jeho fyzická i virtuální adresa, velikost bloku a hexadecimální reprezentace jeho hodnoty. Z hlediska formátování výstupu je hexadecimální reprezentace hodnoty paměťového bloku zkrácena, pokud přesahuje určitou mez. Toto chování jde potlačit parametrem **-n**.

Dalším důležitým parametrem je **-b**, který způsobí vypsání informací pouze o paměťovém bloku, ze kterého byla zkopírována hodnota do registru *EIP*. Podobný účel plní parametr **-p**, který vypíše hexadecimální reprezentaci hodnoty síťového paketu, který obsahoval data, která byla následně překopírována do registru *EIP*.

Výše zmíněné parametry lze libovolně kombinovat. K vypsání všech možných informací o útoku slouží parametr **-a**.

7.4 argos2mysql

Nástroj *argos2mysql* zpracovává data generovaná nástrojem Argos a ukládá je do databáze. K vlastní extrakci dat z log souborů je použita výše zmíněná aplikace *logview*.

7.4.1 Zvolené technologie

Na rozdíl od předchozí aplikace je aplikace *argos2mysql* napsána v jazyce *Python 2.6*. Důvodem je pohodlnější a rychlejší vývoj oproti jazyku C. Pro jazyk C hovoří především rychlost výsledné aplikace, avšak ta v tomto případě nehraje nejdůležitější roli. Jako databázový systém byl zvolen *MySQL 5*. Důvodem je jeho rozšířenost a snadná použitelnost. Pro práci s MySQL v jazyce Python je použita knihovna *mysql-python 1.2.3*. Z dalších externích modulů pro Python je použit modul *pynotify*, který je v aplikaci použit pro detekci nově vytvořených log souborů ve sledovaném adresáři.

7.4.2 Popis aplikace

Hlavním úkolem aplikace je zpracování log souborů generovaných nástrojem Argos a následné ukládaní extrahovaných informací do databáze. K tomuto slouží hlavní tři módy programu. Aplikaci může být přes argument **-f** předán jednotlivý soubor obsahují záznam o útoku a ten je následně zpracován.

Další volbou je parametr **-d**, přes který je předáno jméno adresáře, ve kterém jsou poté vyhledány všechny Argos log soubory a ty jsou sekvenčně zpracovány.

Posledním a nejdůležitějším módem aplikace je tzv. *monitor mód*. Tento mód se aktivuje parametrem **-m**, kterému je předán název adresáře, který bude monitorován. Program následně tuto složku sleduje a v momentě výskytu nového log souboru soubor zpracuje.

Díky využití modulu **pynotify**, je použito pasivní čekání a aplikace tak při čekání na nový soubor zbytečně nespotřebovává procesorový čas.

Monitorovací mód je nejdůležitějším módem této aplikace z hlediska automatického zpracování. Umožňuje tak následující aplikaci: Argos je spuštěn a připraven zaznamenávat útoky. Výstupní soubory jsou ukládány do určitého adresáře. Nad tímto adresářem je spuštěn monitorovací mód aplikace **argos2mysql**. Jakmile poté dojde k útoku, který je Argosem zachycen, Argos vygeneruje soubor se záznamem tohoto útoku, **argos2mysql** tento soubor zaregistruje, použije aplikaci **logview** k extrakci dat z log souboru a výsledné informace uloží do databáze. To vše bez nutnosti zásahu člověka. Data jsou tedy ve výsledku uložena v databázi, odkud mohou být zpětně vyzvednuta a podrobena další analýze. Ta už je však mimo rozsah této práce. Činnost programu v tomto módu je zobrazena na výpisu 7.3

Výpis 7.3: Argos2mysql: Výpis nápovědy

```
$ ./argos2mysql.py -m ../argos_image/
[19:36:57] Monitor mode on ../argos_image/
[19:38:24] ~/argos/argos.csi.1951951906
            Processing file...
[19:38:25] ~/argos/argos.csi.1951951906
            File successfully parsed and saved to database
[19:39:04] ~/argos/argos.csi.717528971
            Processing file...
[19:39:04] ~/argos/argos.csi.717528971
            File successfully parsed and saved to database
```

Dalšími dodatečnými vlastnostmi aplikace jsou parametry **-t** a **-c**. Parametr **-c** aktivuje vygenerování potřebných tabulek a závislostí v zadané databázi. Ta bude poté použita pro ukládání extrahovaných dat. Parametr **-t** se pak týká také této databáze a jeho funkcí je vymazání dat ze všech tabulek s daty. Jinými slovy po použití tohoto parametru se databáze dostane do stejného stavu, jako byla po prvním použití parametru **-c**.

Poslední pomocnou funkcí aplikace **argos2mysql** je extrakce hexadecimální reprezentace síťového paketu, který způsobil útok[2] a jeho následné převedení do formátu *pcap*. Formát pcap[3] je rozšířený formát pro uložení síťových dat. První knihovna *libpcap* byla vytvořena autory známého nástroje **tcpdump**[4], nyní tento formát využívá množství aplikací v čele se síťovým analyzátorem **Wireshark**[5]. Tato funkce je výhodná především proto, že daný paket lze po převedení do *pcap* formátu jednoduše analyzovat například již ve zmíněném nástroji **Wireshark**. Výpis nápovědy aplikace **argos2mysql** shrnující všechny funkce programu je na výpisu 7.4.

Výpis 7.4: Argos2mysql: Výpis nápovědy

```
$ ./argos2mysql.py -h
Argos2mysql Tool v1.0
```

[2] Přesněji řečeno přinesl do počítače data, která byla následně nahrána do registru EIP, načež došlo k jejich nelegálnímu použití.

[3] http://www.tcpdump.org/pcap3_man.html

[4] http://www.tcpdump.org

[5] http://www.wireshark.org

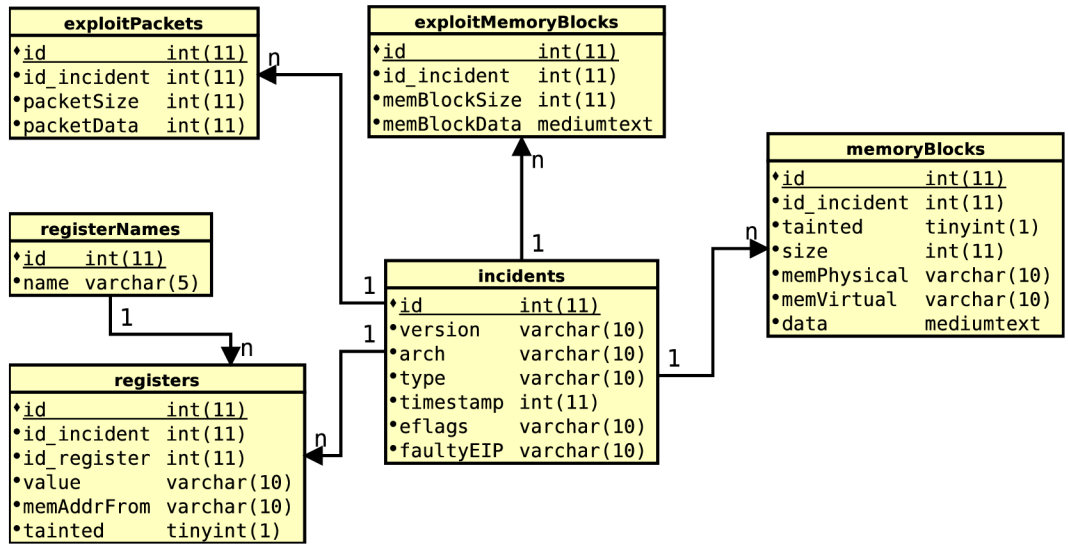
```
Author: Lukas Antal
Usage: argos2mysql.py [-m|-d|-f|-p name] [-c database] [-ht]
    -m   Monitor specified directory
    -d   Process all Argos logs in specified directory
    -f   Process specified Argos log file
    -p   Export exploit packet from specified attack log
         to pcap format
    -t   Truncate all tables in argos-related database
    -c   Create database and prepare all tables
    -h   Print this help
```

Pro korektní provedení akcí specifikovaných parametry **-m, -d, -f, -t** a **-c** je nutné připojení k databázi. Přihlašovací údaje jsou specifikovány v konfiguračním souboru *argos2mysql.ini*, kde je nutné vyplnit identifikaci počítače, na kterém databáze běží, přihlašovací údaje k databázi a název databáze. Struktura tohoto souboru je zobrazena na výpisu 7.5.

Výpis 7.5: Argos2mysql: Konfigurační soubor

```
[db]
host = localhost
user = dbuser
password = dbpassword
db = argos
```

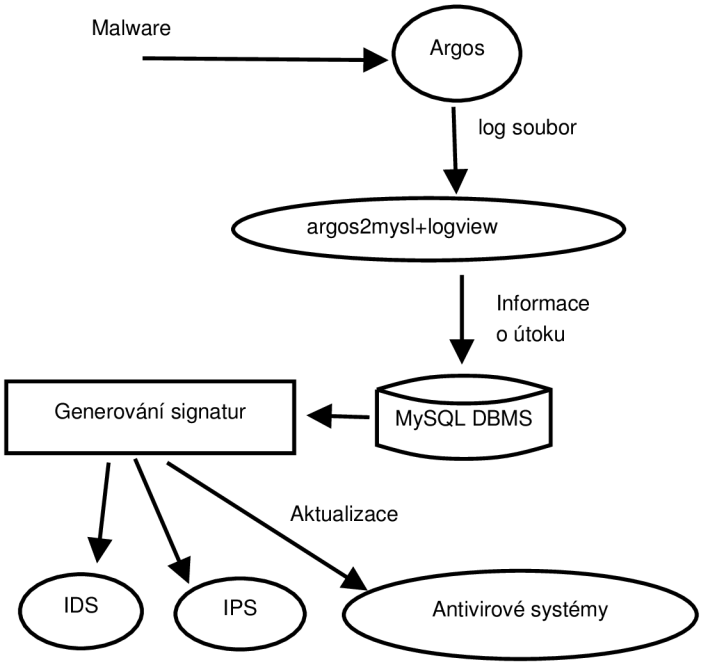
Jak již bylo výše zmíněno, aplikace používá databázový systém *MySQL*. Data ukládá do celkem šesti tabulek propojených primárními a sekundárními klíči. Databázová struktura je znázorněna na obrázku 7.2.



Obrázek 7.2: argos2mysql: Schéma ukládaných dat

7.5 Vyhodnocení výsledků

Schéma zapojení aplikací implementovaných v rámci této práce vzhledem k automatizovanému zpracování útoků je znázorněno na obrázku 7.3. Aplikace **logview** a **argos2mysl** implementovány v rámci této práce tvoří důležitý prvek v architektuře automatizovaného zpracování útoků s využitím nástroje typu shadow honeypot. Tyto implementace automatizovaly ukládání dat z útoků do databáze. Data z útoku jsou v databázi připravena pro další zpracování téměř okamžitě po zaregistrování útoku. Na tento postup pak může navázat další zpracování těchto dat, které bylo popsáno výše.



Obrázek 7.3: Schéma zachycující architekturu pro automatizované zpracování útoků s využitím výsledků této práce.

Kapitola 8

Závěr

8.1 Zhodnocení výsledků práce

Tato práce měla za cíl zmonitorovat současný stav nástrojů typu shadow honeypot. Shadow honeypot je zcela nová architektura honeypotů, která je zatím rozebírána spíše v teoretických pracích. Přesto už postupně vznikají první prototypy implementací. Mezinárodní projekt *NoAH* (viz kapitola 4) je právě jedním z takových projektů, jež si dal jako jeden z cílů vyprodukovat funkční prototyp nástroje typu shadow honeypot. Právě na tento projekt a jeho výsledné implementace jsem se v práci zaměřil.

Cílem bylo vybrat implementaci shadow honeypotu, která by byla nejvhodnější z hlediska automatizace zpracování kyberútoků. Z dostupných výsledných implementací projektu byly pro analýzu vybrány dva nástroje. Prvním byl klientský honeypot **Honey@Home** (viz kapitola 5), který však nesplňoval některé z důležitých podmínek a navíc se nepodařil zprovoznit na žádném z testovacích strojů pod operačním systémem Windows ani Linux.

Druhým nástrojem vybraných k důkladnější analýze byl nástroj **Argos** (viz kapitola 6). Tentokrát již plnohodnotný *shadow honeypot* s dostupnými zdrojovými kódy. Nástroj je založen na emulačním software **QEMU**, nad který přidává vrstvu pro detekci útoků. Argos je schopný zachytit široké spektrum útoků a to jak na jednotlivé aplikace, tak i přímo na jádro operačního systému. Hlavními metodami útoku, které Argos dokáže rozpoznat, jsou buffer overflow, zneužití chyby formátovacího řetězce a podvržení parametrů funkcí systémových volání.

Důležitou vlastností tohoto nástroje je schopnost útok důkladně zaznamenat do výstupních souborů. Tyto binární soubory pak obsahují informace o registrech či paměťových blocích, které se účastnily daného útoku. Argos navíc sleduje veškerá data pocházející ze sítě a k útoku je tak možné dohledat konkrétní síťové pakety, které nesly škodlivá data (exploit a shellcode).

Jelikož hlavním cílem práce je automatizované zpracování útoků, v praktické části jsem navrhnul a implementoval řešení pro zpracování dat z výstupních souborů. Toto řešení jsem rozdělil na dvě spolupracující aplikace **logview** a **argos2mysql**. První zmíněná má za úkol zpracovat informace z výstupního souboru nástroje Argos do textové podoby. Jedná se o informace týkající se stavů registrů procesoru v době útoků, zúčastněných paměťových bloků či přímo o síťovém paketu nesoucím škodlivá data. Tyto informace následně aplikace vytiskne na standardní výstup. Druhým nástrojem pro zpracování výstupů je program nazvaný **argos2mysql**. Ten ke své činnosti využívá výše zmíněný **logview**. Cílem tohoto nástroje je tak ukládání dat z **logview** do databáze. Program podporuje několik módů z nichž nejdůležitější je tzv. *monitorovací režim* nad určenou složkou. Do této složky

ukládá Argos své výstupy. Jakmile se v adresáři objeví nový soubor, **argos2mysql** ho okamžitě zaregistruje, zavolá **logview** pro jeho převod na textová data, ta zpracuje a uloží do databáze.

Z databáze jsou pak data z útoků lehce dostupná pro další zpracování. Tím snad nejdůležitějším je generování pravidel a signatur z útoků. Ty jsou poté následně distribuovány jako aktualizace IDS/IPS a jiným anti-malwarovým nástrojům, které se starají o ochranu sítě a koncových stanic. Tato další zpracování jsou však už mimo rozsah této práce.

Cíle vytyčené v úvodu práce se podařilo naplnit. Byla vybrána vyhovující aplikace typu shadow honeypot, nad kterou bylo implementováno zpracování jejích výstupů do databáze.

8.2 Budoucí vývoj

Rizika spojená s hrozbou malware jsou stále větší a současné systémy pro detekci a rozpoznání malware přestávají být účinné a to hlavně z důvodu velmi omezené detekce tzv. *zero-day* útoků. Nutnost zavést do této oblasti nové principy a techniky je tak naprosto nevyhnutelná. Tato práce se zabývala jedním s možných řešení, a to architekturou **shadow honeypot**. Podle mého názoru se jedná o velice perspektivní návrh řešení detekce *zero-day* útoků, který ač je teprve v počátcích, má velice velké šance stát se v budoucnu jedním z hlavních pilířů této oblasti. Rozšíření této koncepce do komerčních anti-malware nástrojů nastane dle mého názoru do několika málo let.

Literatura

[1] Anagnostakis, K. G.; Sidiroglou, S.; Akritidis, P.; aj.: Detecting targeted attacks using shadow honeypots. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, Berkeley, CA, USA: USENIX Association, 2005, s. 9–9.

[2] Bayoglu, B.; Sogukpinar, I.: Polymorphic worm detection using token-pair signatures. In *SecPerU '08: Proceedings of the 4th international workshop on Security, privacy and trust in pervasive and ubiquitous computing*, New York, NY, USA: ACM, 2008, ISBN 978-1-60558-207-8, s. 7–12.

[3] Costa, M.; Crowcroft, J.; Castro, M.; aj.: Vigilante: end-to-end containment of internet worms. In *SOSP*, 2005, s. 133–147.

[4] Crandall, J. R.; Chong, F. T.: Minos: Control Data Attack Prevention Orthogonal to Memory Model. *Microarchitecture, IEEE/ACM International Symposium on*, ročník 0, 2004: s. 221–232, ISSN 1072-4451.

[5] Grégio, A.; Santos, R.; Montes, A.: Evaluation of data mining techniques for suspicious network activity classification using honeypots data. Brazilian Institute for Space Research, Av. dos Astronauta, 1758, 2007.

[6] Leder, F.; Werner, T.: Know Your Enemy: Containing Conficker. [online], 30.3.2009, the Honeynet Project, [cit. 27.01.2010].
URL http://p3security.org/files/KYE-Conficker.pdf

[7] Newsome, J.; Song, D.: Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS)*, 2005.

[8] Ogorkiewicz, M.; Frej, P.: Analysis of Buffer Overflow Attacks. [online], 8.11.2002, [cit. 02.03.2010].
URL http://www.windowsecurity.com/articles/Analysis_of_Buffer_Overflow_Attacks.html

[9] PandaLabs: Annual Report 2009. [online], 2010, [cit. 22.01.2010].
URL
http://www.pandasecurity.com/img/enc/Annual_Report_PandaLabs_2009.pdf

[10] Portokalidis, G.; Slowinska, A.; Bos, H.: Argos: an Emulator for Fingerprinting Zero-Day Attacks. In *Proc. ACM SIGOPS EUROSYS'2006*, Leuven, Belgium, April 2006.

[11] Ross, J.: Malware Analysis for the Enterprise. [online], 2010, black Hat DC 2010, [cit. 12.03.2010].
URL http://www.blackhat.com/presentations/bh-dc-10/Ross_Jason/Blackhat-DC-2010-Ross-Malware-Analysis-for-the-Enterprise-wp.pdf

Příloha A

Obsah přiloženého CD

./argos2mysql/

Tento adresář obsahuje zdrojové soubory skriptu argos2mysql, který slouží pro ukládání dat z útoků do databáze. Nápovědu programu je možné si nechat vypsat příkazem *./argos2mysql.py -h*.

./argosLogFiles/

Adresář argosLogFiles obsahuje výstupní soubory nástroje Argos (z aktivovaný NetTracker módem) ze tří útoků za použití exploitu využívajícího zranitelnost MS08-67.

./bp/

Tento adresář obsahuje práci v elektronické podobě ve formátu pdf. V podadresáři *src* se pak nachází zdrojový text.

./logview/

Adresář *logview* obsahuje stejně pojmenovaný nástroj sloužící na převod binárních výstupních souborů nástroje Argos na textová data. V adresáři je jak binární ELF soubor, tak i zdrojové kódy spolu se souborem *Makefile*. Nápovědu programu je možné si nechat vypsat příkazem *./logview -h*.