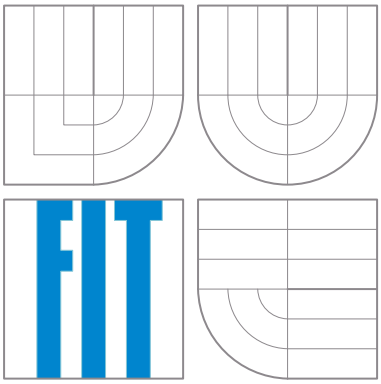

Dokumentace k projektu pro předmět ISA

FTP/SSH Honeypot

20. listopadu 2015

Autor: Dominika Regéciová, xregec00@stud.fit.vutbr.cz

Obsah

1 Úvod

Honeypot je označení pro informační systém, jehož účelem je přitahovat potencionální útočníky a zaznamenávat jejich činnost (Wik, 2015b).
Honeypot simuluje reálný systém a automaticky detekuje, zda je pod útokem (například malwaru). Tento způsob je rychlejší a bezpečnější, než sbírání dat ze skutečných funkčních systémů, které byly již napadeny.
Cílem naší aplikace je vytvoření Honeypotu, který bude schopen simulovat reálný FTP, či SSH server, dle nastavení. V obou módech bude server žádat autentizaci uživatele, kterou však bude odmítat a získané údaje vždy zaznamená do souboru.
FTP/SSH Honeypot je tzv. **low-interaction honeyspot** (Wik, 2015a), což znamená, že implementuje pouze část služeb reálného serveru, které zajistí sběr dat o přihlašovaných uživatelích.
V následujících kapitál si vysvětlíme základní informace o protokolech FTP [3] a SSH [4] , představíme si návrh aplikace [2] se zajímavými implementačními detaily a představíme si stručný návod, jak aplikaci používat [5] .
Netrpělivý čtenář může také využít souboru README, jenž obsahuje základní informace pro nastavení a spuštění serveru.
V předposlední kapitole si zhodnotíme dosažené výsledky [6] .
Veškeré použité zdroje jsou uvedeny v závěrečné kapitole Literatura [6].

2 Návrh aplikace

Logická struktura projektu FTP/SSH Honeypotu je rozdělena na 4 části - řídící část, FTP server, SSH server a logování.
Řídící část přijímá parametry spouštění, které zkontroluje a dle nich rozhoduje o spuštění FTP, či SSH serveru.
Poslední část se pak stará o zaznamenávání dat do logovacího souboru a vypisování nápovědy či chybových hlášení.
Pro implementaci byl zvolen jazyk C++, avšak bez využití technik OOP.

2.1 Implementace

Souborová struktura odráží logické členění. Řídící část se nachází v souborech *fakesrv.cpp* a *fakesrv.hpp*.
Simulace serverů se spouští v *ftp.*pp* a *ssh.*pp*.
Logování pak v *logging.*pp*.
Projekt je doplněn o Makefile, README a tuto dokumentaci.
Důležité podotknout, že získávání aktuálního času a převod do požadované podoby byl inspirován kódem dle elektronického článku (Kalev, 2003).

3 FTP

3.1 Protokol FTP

FTP, neboli **File Transfer Protocol**, slouží k sdílení a přenosu souborů přes vzdálené počítače napříč sítí.
Pro přenos využívá spojované transportní služby TCP.
Definici nalezneme v RFC 959 (J. Postel, 1985), kde nás bude zajímat především navázání spojení a způsob autentizace.
Protokol navazuje hned dvě spojení. Řídící na portu 21 a datové na portu 22, které obstarává práci s přenášenými soubory.
Nás z pochopitelných důvodů bude zajímat pouze spojení řídící.
Nutné zmínit, že protokolu chybí šifrování uživatelského jména a hesla, což vede ke snížení bezpečnosti.

Jak tedy připojování k FTP serveru probíhá?
Komunikaci zahajuje klient s žádostí o spojení. Pokus na serveru již není připojeno příliš mnoho klientů, server odpoví vítací zprávou `220 Service ready for new user.`
Poté je vyžadováná autentizace uživatele.
Klient zašle uživatelské jméno v podobě `USER name`.
Náš server poté odešle výzvu pro heslo: `331 User name okay, need password.`
Heslo klient zasílá zprávou ve tvaru: `PASS xxx`, na kterou však dostane vždy stejnou odpověď:
`530 Not logged in.`, místo `230 User logged in, proceed.`, které by značilo úspěšné přihlášení.
Poté server ukončí spojení.
Záznam s režimem, časem, ip adresou klienta a jeho přihlašovacími údaji je poté přidán do logovacího souboru.
Server také rozeznává požadavek QUIT na ukončení spojení ze stranu klienta.

3.2 Implementace

Server FTP má být dle zadání implementován pomocí BSD.
Uvažujeme konkurentní server, tedy se současným přístupem více uživatelů, avšak omezeným počtem. Pro implementaci byla zvolena vlákna skrze knihovnu `libpthread`. Protože však máme i společný sdílený soubor, bylo nutné přidat zamykání přístupu pomocí `pthread_mutex_t`.
Nakonec je s mutexem řešen i maximální počet klientů, kteří jsou obsluhovány serverem v jeden okamžik. Mutex ve vhodný čas inkrementuje, či dekrementuje globální celočíselnou proměnnou. Pokud je na server připojeno příliš mnoho klientů, dalším není umožněno navázat spojení.
Největší ošemetností byla podpora IPv4, tak IPv6 protokolů. Tato skutečnost se musela kontrolovat na vícero místech, protože i přes delší existenci protokolu IPv6 stále neexistují funkce, které by zvládaly obě verze.
Také získávání adresy uživatele byl trochu oříšek, protože taktéž nemusela být v 32-bitovém tvaru.
Proto je nejdříve uložena do `sockaddr_storage`, přes funkci `getpeername` je zjištěna rodina (`AF_INET`, či `AF_INET6`) a dle toho je pak adresa přeložena do textové podoby, protože je potřeba pouze pro logovací soubor, nikoliv již pro komunikaci.
Inspirace: část projektu byla inspirována projektu do IPK pro rok 2014, xregec00. Jednalo se o jednoduchou implementaci serveru a klienta pro přenos souborů na TCP, avšak pouze pro IPv4. Protože však nešlo o přímé převzetí kódu, ve zdrojových souborech není označení, jak je tomu v případě Get_Time().
Pro oživení významu funkcí skvěle posloužila prezentace (Jaroslav Ráb, 2012) a v případě potřeby bližších informací `man` nápověda.

4 SSH

4.1 Protokol SSH

Jak jsme si uvedli již v kapitole [3] – Protokol FTP, autentizace přes FTP probíhá bez šifrování.
Ne vždy je však vhodné, aby heslo a další citlivé údaje procházeli nezabezpečenou sítí v nezašifrované podobě. Právě proto existuje protokol SSH **Secure Shell**, který zajišťuje silné šifrování, šifrovací autentizaci počítačů, a ochranu integrity. (Ylonen, 2006)
Stejně jako FTP, i SSH pracuje nad TCP a server standardně naslouchá na portu 22.
Na SSH server je možné se přihlásit pomocí páru privátního a veřejného klíče. Již to stačí k získání přístupu, avšak většina serverů vyžaduje uživatelského jméno a heslo pro větší bezpečnost.
Architektura SSH je rozdělena na více částí, nás bude zajímat pouze `transport layer`, která se stará o výměnu klíčů, v našem případě RSA klíčů, serverovou autentizaci, nastavení šifrování, komprese a verifikaci integrity.

4.2 Implementace

Implementace SSH serveru je velmi podobná FTP implementaci. Taktéž je zde použita knihovna `libpthread`. zápis do souboru, počet klientů je opět řešen přes `pthread_mutex_t`.
Hlavním rozdílem je využití knihovny `libssh`. Ta řeší spoustu věcí, včetně navázání spojení a komunikací s klientem, avšak je důležité porozumět tomu, jaké funkce je potřeba použít.
Dokumentace není příliš nápomocná a nejjednodušší je asi hledat informace přímo v implementaci, kde lze nalézt i užitečné příklady.
Změnou je také skutečnost, že klienti mají více pokusů na zadání hesla, což je zohledněno ve funkci `Authentication`. Podle počtu pokusů poté server odpojí pomocí `ssh_disconnect`.
Komunikace jinak větší míry řeší knihovní funkce, kde vlastně pouze zjišťujeme a nastavujeme type a subtype zprávy.
Zrádností SSH varianty byl timeout pro příjem zpráv. Ten při pokusech způsobuje občasné dřívější odpojení klientů, ale protože v zadání nebylo specifikované, že bychom měli tuto vlastnost měnit, není timeout přenastavován.

5 Návod na použití

Aplikaci lze stáhnout v souboru *xregec00.tar.gz*, který lze extrahovat pomocí příkazu `tar -xzf xlogin00.tar.gz`.
Následně je potřeba projekt přeložit: `make`.
Doporučuje se také použít příkazu `chmod +x fakesrv` pro povolení spouštění aplikace.
Pokud budeme používat SSH server, je nutné vygenerovat RSA klíče, které získáme například s pomocí `ssh-keygen -t rsa -f name.key`.
Dále stačí spustit program.
`./fakesrv -m mode -a address -p port -l logfile -r rsakey -c max_clients -t max_attempts`
Povinnými parametry jsou:

-m pro mód, tedy SSH, či FTP

-a pro adresu ve formátu IPv4 nebo IPv6

-p pro číslo portu

-l pro logovací soubor

Pokud je server nastaven na SSH mód, je nutné zadat cestu k RSA klíči:

-r pro RSA klíč

Zcela volitelné jsou pak parametry:

-c pro maximální počet klientů, které server bude obsluhovat v jeden čas, výchozí hodnota je 10

-t pro maximální počet pokusů při zadávání hesla pro SSH mód, výchozí hodnota je 3

Pro připojení lze využít volně dostupné klienty, nebo se lze jednoduše připojit přes terminál:

```
ftp adresa port
ssh -l login adresa -p port
```

6 Shrnutí

Seznámili jsme se s tím, co je to Honeypot a jaké je jeho využití v sítích. Zopakovali jsme si princip fungování protokolů FTP a SSH, včetně programování síťových aplikací s pomocí BSD socketů a knihovny libssh.
Výsledná aplikace by sice mohla mít elegantnější řešení podpory IPv4/IPv6 protokolů a třeba více využívat schopností jazyka C++ (OOP, vektory,. . .), avšak odzkoušení programu (testování během a po implementaci) ukázalo očekávané chování aplikace a není známa žádná vážnější chyba či nedostatek v běhu aplikace.

Literatura

Honeypot(computing) [online]. 2015a. [cit. 12. 11. 2015]. Dostupné z: https://en.wikipedia.org/wiki/Honeypot_%28computing%29.

Honeypot [online]. 2015b. [cit. 12. 11. 2015]. Dostupné z: https://cs.wikipedia.org/wiki/Honeypot.

J. POSTEL, J. R. *RFC 959: FILE TRANSFER PROTOCOL (FTP)* [online]. 1985. [cit. 12. 11. 2015]. Dostupné z: https://www.ietf.org/rfc/rfc959.txt.

JAROSLAV RáB, O. R. *BSD schránky* [online]. 2012. [cit. 12. 11. 2015]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IPK-IT/lectures/ipk201

KALEV, D. *Formatting Date and Time with strftime()* [online]. 2003. [cit. 12. 11. 2015]. Dostupné z: http://www.informit.com/guides/content.aspx?g=cplusplus&seqNum=340.

YLONEN, T. *RFC 4253: The Secure Shell (SSH) Transport Layer Protocol* [online]. 2006. [cit. 12. 11. 2015]. Dostupné z: https://www.ietf.org/rfc/rfc4253.txt.