Introduzione alla Sicurezza dei Sistemi

(seconda parte)

Gabriele D'Angelo

gda@cs.unibo.it

http://www.cs.unibo.it/~gdangelo/

Università di Bologna

Corso di Laurea in Scienze dell'Informazione

Novembre, 2006

Scaletta della lezione

- Buffer Overflow (seconda parte)
- Modelli per l'Access Control
- Intrusion Detection System (IDS) basati su file-system
- Cavalli di Troia
- Vulnerabilità del kernel
- Rootkit: definizione e contromisure
- Security by obscurity
- Altre metodologie tipiche d'attacco
- La sicurezza come processo
- Il problema della disclosure
- Terminologia minima: Hacker vs. Cracker
- Note e bibliografia

Warning!

- Quanto vi presenterò oggi e nelle prossime lezioni è a

  puro scopo didattico

- Le intrusioni nei sistemi informatici sono punite dalla legge, sia civilmente che penalmente, quindi non è il caso di giocare senza capire in modo approfondito cosa si sta facendo e quali sono le conseguenze delle proprie azioni

- Esistono macchine e reti **virtuali**: usiamole!

- Vi impegnate ad usare quanto appreso a solo scopo difensivo?

Effetti collaterali dei buffer overflow

- Esempio banale di vulnerabilità locale: un programma inserisce in un buffer una stringa ottenuta da input **senza controllarne la lunghezza**
- Invocando il programma **con una stringa opportuna** lo si induce ad eseguire codice arbitrario (ad esempio una shell)
- Possono **non** esserci conseguenze negative (salvo l'eventuale segmentation fault): ad esempio se la vulnerabilità è locale e il programma non ha permessi di superutente
- Altre conseguenze:
  - Cosa succede se si tratta di un server in ascolto sulla rete?

    Si ottiene una **shell locale**!
  - Cosa succede se il programma ha permessi di superutente?

    Si ottiene una **shell di root**!

Tecniche di protezione dai buffer overflow

Approccio ottimale: *gestione attenta ed accurata di input e memoria*

Tecniche utili per mitigare il problema:

- **PaX**

  Implementa il principio del **privilegio minimo** applicato alle pagine di memoria. In breve: le aree di memoria che contengono dati vengono rese *non eseguibili*, e quelle relative al programma *non scrivibili*. Inoltre introduce alcune **forme di casualità** (controllata) nella gestione della memoria. Disponibile come patch per il Kernel Linux

- **Stack-smashing protection**

  Un insieme di tecniche per individuare i buffer overflow e prevenire il loro sfruttamento maligno. Esistono varie implementazioni ad esempio StackGuard e SPP (ProPolice). A grandi linee il meccanismo è basato su **valori sentinella all'interno degli stack frame**. Qualora questi valori vengano *alterati* vi è alta probabilità che sia avvenuto un buffer overflow. SPP è una patch da applicare al compilatore GCC

Utilità dei modelli per l'Access Control

Modelli per l'Access Control:

- **Discretionary Access Control (DAC)**

  Implementa il concetto di "*sicurezza attraverso il possesso*". Ad esempio il proprietario di un file può definire (a sua discrezione) i permessi in lettura, scrittura ed esecuzione per quel file. Questo ovviamente si applica anche ai nuovi file

- **Mandatory Access Control (MAC)**

  Le politiche di accesso sono determinate *dal sistema e non dall'utente*. Secondo questo modello esistono delle policy di sistema definite dall'amministratore che *non possono* essere scavalcate dall'utente

- **Role-Based Access Control (RBAC)**

  In questo caso la politica di accesso dipende dalla *funzione* svolta in quel momento

- **Multi Level Security Model:** ("Top Secret", "Secret", "Confidential"...)

Contromisure, dal punto di vista dell'amministratore

- Prevenzione (sistemi sempre **aggiornati,** *con attenzione*)
- Informazione (**conoscere il problema** e *seguirlo nella sua evoluzione*)
- Detection: molto spesso **non ci si accorge delle intrusioni**. Chi si vanta di non aver mai subito un'intrusione con altra probabilità semplicemente non si è mai accorto dell'intrusione
- In alcuni casi la reazione è molto complicata:
  - *Come si mette in sicurezza un sistema compromesso*?
  - Come si può reagire ad uno **zero-day exploit,** ovvero un *exploit che si basa su una vulnerabilità che è appena stata scoperta/annunciata*?

Filesystem-based Intrusion Detection System (IDS) - AIDE

Advanced Intrusion Detection Environment (AIDE)

- È basato su un database che contiene una serie di informazioni su ogni singolo file e directory che fanno parte del file-system
- Lo scopo è quello di verificare l'integrità dei singoli file, *rilevando* e *riportando* eventuali modifiche:
  - ora e data di creazione e modifica
  - dimensione
  - algoritmi di digest (md5, sha1, rmd160, tiger, haval ecc)
- Attraverso espressioni regolari è possibile definire quale parte del file-system considerare e secondo quali modalità
- Il database generato **non è crittografato e nemmeno firmato**

Filesystem-based IDS - Tripwire

- È basato su un database che contiene una serie di informazioni su ogni singolo file e directory che fanno parte del file-system
- Esiste in varie versioni commerciali e libere
- Supporta un meccanismo di firma e crittografia del database generato a partire dei file analizzati, la stessa cosa avviene per i file di configurazione

- È evidente come l'approccio nei confronti della crittografia sia opposto a quello usato da AIDE

- **È sufficiente garantire una buona integrità del database per essere sicuri che il sistema sia integro?**

Filesystem-based IDS - Samhain

- Samhain è un tool simile ad Aide e Tripwire ma che integra il supporto per un repository centralizzato (struttura client-server)
- È basato come i precedenti su checksum crittografici
- Il server è basato su un DBMS (es. PostgreSQL, MySql ecc)
- I client (samhain) si collegano al server (yule) attraverso connessioni crittografate
- Configurazione e database possono essere firmati con OpenPGP, la crittografia è basata su AES
- I client si "autenticano" al server attraverso una "embedded key" per migliorare la resistenza ai trojan, una parte del client lavora a livello kernel

Cavalli di troia

- Spesso le apparenze non corrispondono alla realtà, anche in questo caso il software non fa eccezione
- Chi ci assicura che il software che compriamo o che scarichiamo da Internet non si comporti in maniera diversa da come dichiara o da come ci aspettiamo?
- Software "**closed source**" -> ci fidiamo del **produttore**, del suo buon nome, della sua voglia di mantenersi "**rispettabile**"
- Software "**open source**" -> usualmente ci fidiamo della "**comunità**", teoricamente sarebbe possibile controllare tutto. Potrebbero essere compromessi i binari, il distributore o direttamente i sorgenti (eventualmente anche quelli del kernel)

Vulnerabilità del kernel

- Anche il kernel può essere vulnerabile, nella sua parte monolitica oppure nei moduli. Molto spesso i device driver vengono diffusi sotto forma di moduli. Come valutiamo la sicurezza dei moduli chiusi, forniti da terze parti?
- Esempi di vulnerabilità del kernel:
  - Stack TCP/IP, frammentazione (da **remoto**)
  - Memory management (da **locale**)
- Le vulnerabilità locali vengono spesso *considerate a basso rischio*, è facilmente una valutazione errata. **Privilege escalation**:

vulnerabilità remota (es. www/ftp/mail) ->
accesso **non privilegiato** ->
vulnerabilità locale ->
accesso con **privilegi di root**

Vulnerabilità del kernel: un esempio

- Linux kernel do_mremap VMA limit local privilege escalation vulnerability
  - Versioni vulnerabili: 2.2 fino a 2.2.25, 2.4 fino a 2.4.24, 2.6 fino a 2.6.2
  - http://www.isec.pl/vulnerabilities/isec-0014-mremap-unmap.txt
- La vulnerabilità riguarda una system call non privilegiata, mremap()
- È assente il controllo sul valore di ritorno della funzione do_mremap(), che in condizioni particolari può fallire. Sfruttando alcune condizioni particolari (attraverso un eseguibile “set user id”) può portare all'esecuzione di codice con privilegi di root

Rootkit: livello utente, livello kernel

- **Strumento utilizzato da un intrusore dopo aver violato un sistema:** l'obiettivo è quello di garantirsi l'accesso, soprattutto *mascherando la propria presenza* e le tracce lasciate nel sistema

- I rootkit si occupano di cancellare selettivamente parte dei log e di sostituire alcune utility tipiche con versioni apparentemente uguali ma "addomesticate" (es. opera sui login, processi e log)

- Esistono anche i "kernel rootkit" che lavorano a livello di system call. Vengono messi in opera attraverso moduli del kernel (nuovi o infettando quelli già presenti) oppure modificando direttamente la memoria usata dal kernel (/dev/kmem). *Le normali tecniche di rilevazione per rootkit in questo caso non sono efficaci*!

Rootkit: com'è possibile individuarli?

- Nonostante i rootkit cerchino di nascondere la loro esistenza, è molto difficile rendersi completamente invisibili
- Ci sono alcuni tool che tentano di capire se il sistema è stato compromesso da rootkit noti. Ad esempio, **chkrootkit**
- I rootkit noti sono relativamente facili da rilevare; per gli altri esiste solitamente una serie di comportamenti tipici che possono "tradirli"
- I rootkit cercano di nascondere l'esistenza di alcuni processi, modificando i binari di alcuni programmi (ps, top) o interagendo con /proc/, *questo porta a incoerenze rilevabili*

Security by obscurity

- La **segretezza** rende un sistema **sicuro**?
- Meglio tenere **nascoste** le informazioni su un sistema (ad esempio la sua architettura) oppure una loro diffusione permette un **migliore controllo** e una **risposta più rapida** alle vulnerabilità?
- Il caso della crittografia (*principio di Kerckhoff*): **tutto è noto agli attaccanti ad esclusione della chiave di crittazione** e anche
  "**The enemy knows the system**". (*Claude Shannon*)
- Sarebbe preferibile evitare un approccio **dogmatico**: spesso la *security by obscurity* è inutile (es. algoritmi crittografici), in altri casi può migliorare l'effettiva sicurezza di un sistema

Altri strumenti per l'attacco: conoscere per evitare

- Social engineering: molto spesso è sufficiente **chiedere per ottenere**, magari millantando di essere chi in realtà non si è. *Fidarsi spesso è male* (esempio: finti amministratori che chiedono password). Il **Phishing** e il **Pretexting** sono esempi molto attuali
- Eavesdropping: curiosare in giro può rivelare molte informazioni delicate (ad esempio: pagine www, cartelle temporanee nei sistemi, aree di memoria)
- Backdoor: molti programmatori/sistemisti hanno la pessima abitudine di lasciare qualche ricordo indesiderato. *Quello che sembra segreto non è sempre destinato a rimanerlo*, soprattutto ad occhi interessati, esperti ed attenti (ad esempio: in molti prodotti per il networking si sono trovate backdoor volontariamente inserite dagli sviluppatori)

La sicurezza è un processo continuo

- **La sicurezza non è un prodotto ma un processo**: soprattutto in presenza di scenari complessi (es. ambienti di lavoro formati da molti utenti) ottenere un *livello accettabile* di sicurezza implica un lavoro continuo di **educazione, aggiornamento** e **attenzione**

- La realtà mette di fronte ad una scelta di **compromesso** tra **sicurezza** e **facilità d'uso**. Un sistema troppo complicato e quindi scomodo da utilizzare diviene insicuro perchè le politiche concordate *non vengono rispettate*

- La sicurezza rappresenta un **costo economico** molto tangibile, sia quando tutto funziona correttamente, sia in presenza di incidenti. *Il "costo della sicurezza" deve essere confrontato con quello dovuto (o previsto) per gli incidenti, il valore del sistema e dei dati*

Il problema della “disclosure”

- Quando una nuova vulnerabilità viene scoperta e viene scritto un exploit funzionante ci si può comportare in diversi modi
- A chi viene diffusa l'informazione?
  - **full disclosure**: tutti i dettagli sono immediatamente pubblicati
  - **timed disclosure**: i dettagli sono inizialmente comunicati solo agli sviluppatori del software vulnerabile e, dopo un certo lasso di tempo, sono resi noti al resto del pubblico
  - **limited disclosure / no disclosure**: gli unici informati dell'esistenza e dei dettagli della vulnerabilità sono gli sviluppatori del software o nessuno
- Come già detto: un exploit sconosciuto alla comunità della sicurezza o attualmente senza contromisure disponibili è detto **zero-day exploit**
- C'è un approccio migliore degli altri?

Le parole sono importanti: Hacker vs. Cracker

Hacker vs. Cracker

- **Hacker**: "a person who enjoys **exploring the details** of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn the minimum necessary […]"

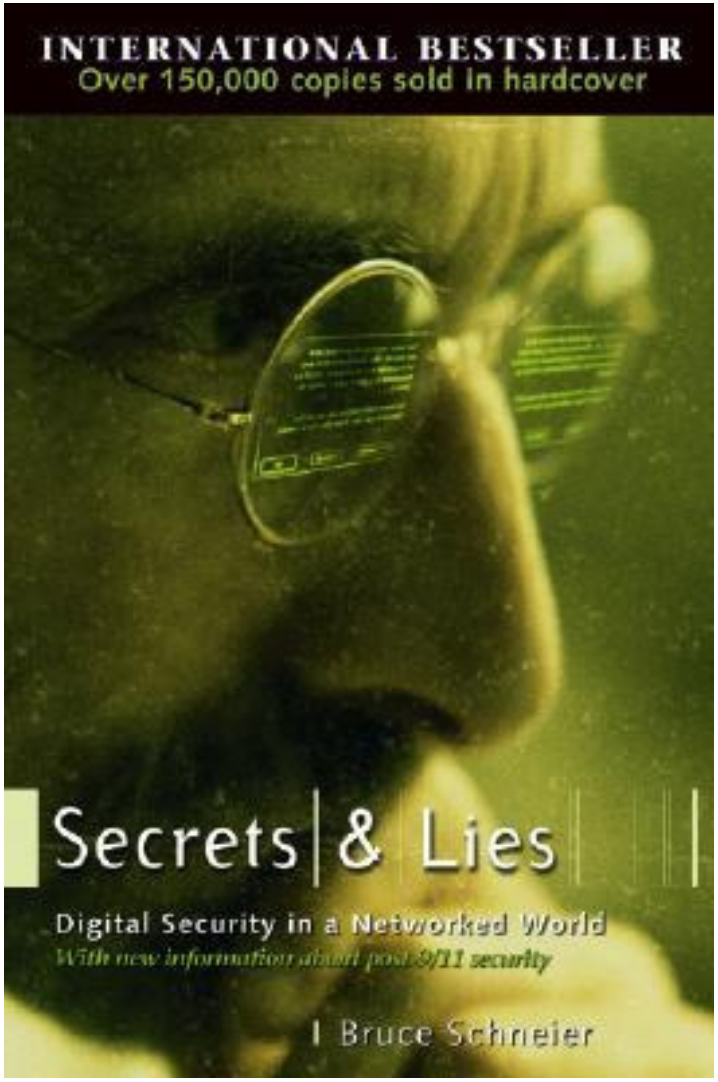
- **Cracker**: "One who breaks security on a system […] Most crackers are only **mediocre** hackers"

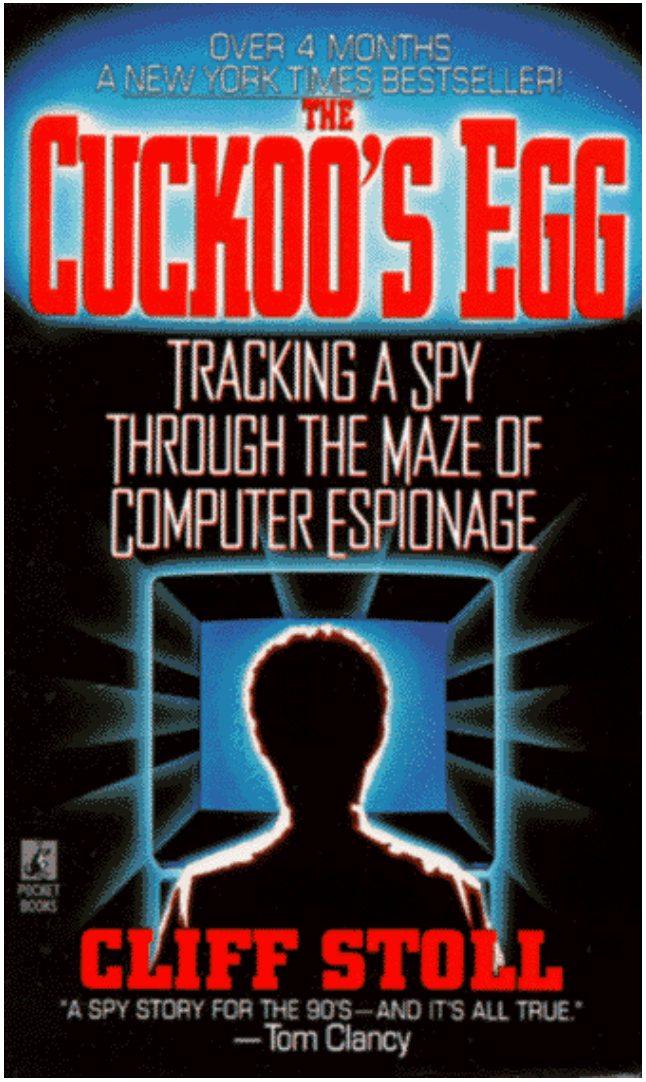
Fonte: The Jargon Dictionary. http://www.catb.org/~esr/jargon/

Note

- Nella realizzazione di queste slide non è stato fatto alcun male a **server reali** con bit reali: solamente **server e reti virtuali**, *si prega vivamente di fare altrettanto!*
- La sicurezza è molto divertente (almeno fino a quando non si viene bucati) ma pone di fronte anche a vari problemi etici. È necessario riflettere anche su questi e non solamente sulla parte prettamente tecnica
- Molte intrusioni non vengono rilevate se non dopo molto tempo o eventualmente mai. Non sei mai stato bucato o non te ne sei mai accorto?
- I tool posso aiutare molto ma a fare la differenza sono comunque la preparazione, l'informazione e le capacità

Testi introduttivi ma estremamente interessanti





Bibliografia

- **The Internet Worm Program: An Analysis.** E.H. Spafford. http://www.textfiles.com/100/tr823.txt
- **Practical Unix and Internet Security.** S. Garfinkel, G. Spafford
- **The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage.** C. Stoll
- **Secrets and Lies. Digital Security in a Networked World.** B. Schneier
- **Schneier on Security.** http://www.schneier.com/blog/
- **BugTraq.** http://www.securityfocus.com/archive/1
- **SecurityFocus.** http://www.securityfocus.com/
- **Open Source Vulnerability Database.** http://www.osvdb.org/
- **Debian Security Information.** http://www.debian.org/security/