Politecnico di Milano
Dip. Elettronica e Informazione
Milano, Italy

Programmazione sicura: introduzione

Ing. Stefano Zanero

15/05/2006

Informazioni utili

- ***Stefano Zanero***
  tel. 4010 - zanero@elet.polimi.it
- Materiale e approfondimenti
  - Slide :-)
  - R. Anderson, "Security Engineering", Wiley: cap 4 (in particolare 4.4),6,18 (in particolare 18.4)
  - Hoglund, McGraw "Exploiting Software", Addison-Wesley
  - Howard, LeBlanc, "Writing Secure Code", Microsoft
- **Di cosa ci occuperemo: aspetti pratici della sicurezza delle applicazioni**
  - **Programmazione sicura**
  - **Cosa significa "exploit"**
  - **Codice virale e maligno**

I tre obiettivi cardine

- ***Confidenzialità***: solo le persone autorizzate possono accedere alle informazioni
- ***Integrità***: solo le persone autorizzate possono modificare le informazioni, e solo nelle modalità per cui sono state autorizzate a procedere.
- ***Disponibilità***: le persone autorizzate possono sempre ricevere le informazioni di cui hanno bisogno entro un tempo "ragionevole" (secondo i requisiti)
- Paradigma "CIA" (Confidentiality, Integrity and Availability)

Implementazione

- Di solito implementiamo questi requisiti mediante
  - Identificazione di **utenti**
  - **Identificazione di risorse**
  - **Creazione di regole d'accesso**
- **Come si pensano queste regole? Quali algoritmi e sistemi si usano per applicarle?**
  - **Lezioni del prof. Paraboschi**
- **Come si implementano queste regole ?**
  - **A livello di infrastruttura: lezioni del corso di "Sicurezza degli Impianti"**
  - **A livello software: le regole sono implementate dalle applicazioni e dal sistema operativo**

Alcuni tipi di attacco (1)

- **Attacchi DOS** (Denial of Service)
  rendere *indisponibili* uno o più servizi erogati da un sistema a causa di una sua saturazione provocata in modo artificiale
  - Flooding e DDOS (Distributed DOS) *intasandone completamente le risorse*
  - DOS mirato a un servizio critico *facendolo crollare*
- **Penetrazione nel sistema**: acquisizione di privilegi non consentiti (**root**) per
  - **Sottrazione** di informazioni (piani industriali, brevetti… alto valore!)
  - **Utilizzo improprio** del sistema (distribuzione materiale illegale, attacchi verso terzi, …)
- **Social Engineering**: metodologia di intrusione che utilizza un approccio non-tecnico, prevalentemente basato su interazioni umane

Alcuni tipi di attacco (2)

- Sniffing: lettura "abusiva" di tutti i pacchetti che transitano in rete mediante sniffer
- Spoofing: falsificazione dell'indirizzo IP del mittente al fine di fingersi "un altro"
- Hijacking: persone non autorizzate prendono il controllo di un canale di comunicazione; la connessione viene utilizzata da una terza parte
- Infezioni da parte di virus & worm

Tecniche di base della sicurezza

- **Identificazione e Autenticazione**: individuazione di un utente e verifica della sua identità tramite:
  - ciò che si sa (condivisione di un segreto, *pwd, …*),
  - ciò che si possiede (*smart-card*, *certificato digitale*,…)
  - ciò che si è (*impronte digitali, iride, …*)
- **Autorizzazioni e controllo accessi**: assegnazione dei diritti di accesso alle risorse (archivi, file, …). Si verifica che un soggetto utilizzi soltanto le risorse alle quali è stato autorizzato (*Access Control Lists, ACL*) ed effettui solo le operazioni permesse
- **Uso di tecniche crittografiche**: utilizzo di algoritmi di cifratura (per rendere non intelleggibile l'informazione), di hashing (per garantire l'integrità)

Sicurezza a livello di OS (1)

- Sistemi operativi progettati per ambienti multi-utente
  - Es: Un*x, Windows 2000/XP,…
- Dotati di meccanismi di autenticazione degli utenti (login)
- Accesso alle risorse e privilegi controllati a livello del sistema operativo
- Pensati anche per l'utilizzo da remoto (*servizi di rete*)
- L'identificazione degli utenti è fornita (tipicamente) da LOGIN e PASSWORD
- Ogni utente appartiene a uno o più GRUPPI
- Ogni file e ogni dispositivo (connessioni, periferiche, ecc.) ha un insieme di permessi che ne regolano l'uso

Sicurezza a livello di OS (2)

- Un primo livello di sicurezza consiste nel garantire che chi accede al sistema sia "riconoscibile"
- Un secondo livello di sicurezza consiste nel garantire che possa accedere solo a file che gli sono permessi
- Unica eccezione l'utente "**`root`**" che per definizione non rispetta nessun permesso.
- I "demoni", o programmi residenti che offrono servizi di rete, rispettano anche loro queste restrizioni, in base ai privilegi degli utenti con cui vengono eseguiti (nel mondo Un*x, SUID: Saved User ID)
- Alcuni demoni devono essere eseguiti SUID **`root`** (cioè senza nessuna restrizione di accesso) per una serie di motivi

Cosa significa “exploit” ?

- Prima idea “ingenua” di attacco a un sistema: indovinare le password (*brute force attack*)
- E’ proprio vero che *“Non è possibile “accedere” al sistema senza fornire login e password”*?
- I “demoni” offrono servizi di rete, a volte anche ad utenti non autenticati.
- Ricordiamoci che i “demoni” usano per le restrizioni il SUID, che a volte è “root”. Quindi un demone può avere un accesso non ristretto al sistema.
- Se si riesce a “imbrogliare” un demone e a fargli eseguire comandi al posto nostro, si può sfruttare il suo SUID e violare le restrizioni d’accesso (tipico meccanismo degli EXPLOIT)

Processi in ambiente UNIX

- Ogni esecuzione di programma in ambiente UNIX dà luogo a un PROCESSO identificato da
  - Un PID, Process Identifier
  - Parametri di scheduling: una policy di scheduling predefinita, oppure i parametri “nice”, “priority” (-20 RT; 20 idle), e “counter” per i processi “SCHED_OTHER”
  - Limiti di risorse per processo
  - “Filesystem root”: il punto da cui il processo pensa che il filesystem cominci
  - La “umask” (set di bit del controllo d’accesso) che sarà attribuita ai file creati dal processo
- Creati con una fork()

UID: da utente a numero

- User e Group vengono mappati a numeri interi denominati UID e GID. UID 0 = root, tutti i permessi garantiti
- Ogni processo contiene informazioni di questo tipo:
  - RUID, RGID: UID e GID dell'utente per cui il processo sta venendo eseguito. In aggiunta, una lista dei gruppi "addizionali" in cui l'utente partecipa.
  - EUID, EGID: UID e GID effettivi per tutti i controlli di privilegio, tranne quelli legati al filesystem
  - FSUID, FSGID: UID e GID usati per i controlli di privilegio su filesystem (solo Linux, gli altri usano UID)
  - **SUID, SGID: UID e GID salvati, utilizzati per "attivare" e "disattivare" privilegi (non tutti gli UNIX)**

SUID: utilizzo

- Alcuni demoni e alcuni programmi devono poter essere eseguiti da chiunque, e tuttavia necessitano di particolari privilegi d'accesso
  - Il caso più evidente è quello dei demoni che devono fare bind() sotto la porta 1024, e sono quindi SUID root
- In questo caso il programma parte con EUID = RUID dell'utente che lo esegue, poi imposta l'EUID al valore del SUID, effettua le chiamate per cui necessita dei privilegi, e in seguito torna al RUID iniziale. Questo procedimento di acquisizione e perdita dei privilegi è fondamentale per la sicurezza.

Cenni sui processi in ambiente NT

- In ambiente NT la struttura dei processi è diversa e per certi versi più complessa
  - Anche qui esiste un PID
  - Esistono delle politiche di scheduling
  - Limiti di risorse per processo
- I processi NT hanno un unico UUID (Unique User ID) con cui vengono eseguiti
- In base a determinate politiche impostate dal kernel possono switchare utente.
- Anche in questo caso esistono delle parti di codice eseguite con privilegi elevati.

Facciamo un esempio

- Prendiamo il caso di un programma SUID root che usi un file di configurazione in formato testo
- Il file può essere "passato" da linea di comando:

```
./programma --config /etc/my-config
```

- Supponiamo ora che un aggressore abbia ottenuto i privilegi di utente sulla macchina, ma non quelli di root...

È molto semplice sbagliare...

ALGORITMO 1

- Start
- **EUID -> SUID**
- Leggi config
- Parsa config
- IF OK continua ELSE scrivi errore
- END

```
[user@host]$ ./programma
  --config /etc/shadow

ERROR in config file,
  line 1:

root:<password>:….
```

ALGORITMO 2

- Start
- Leggi config
- **EUID -> SUID**
- Parsa config
- IF OK continua ELSE scrivi errore
- END

```
[user@host]$ ./programma
  --config /etc/shadow

ERROR: config file not
  found, or unable to
  read
```

Politecnico di Milano
Dip. Elettronica e Informazione
Milano, Italy

Programmazione sicura: nozioni base

Programmazione sicura: what?

- Cosa deve essere programmato in modo sicuro ?
  - Non tutto, nessuno avrebbe mai il tempo !
  - Applicazioni (es.: browser) che attingono e visualizzano dati remoti di sorgente non affidabile
  - Applicazioni utilizzate dal root
  - Demoni, locali o remoti
  - **Applicazioni web**, CGI scripts
  - Applet
  - Programmi SUID
- Ottimo il secure-programming-HOWTO di David Wheeler: http://www.dwheeler.com/secure-programs

Programmazione sicura: how ?

- Ridurre al minimo i programmi che necessitano di essere "SUID" o di avere i privilegi
- KISS (Keep It Simple, Stupid): i programmi privilegiati devono essere semplici
- Le parti di programma in cui vengono effettivamente utilizzati i privilegi devono essere ridotte al minimo
- Appena possibile, il programma dovrebbe perdere definitivamente i privilegi
- Open Design: il programma non deve essere sicuro solo se l'attaccante non lo conosce (come il principio di Kerchoffs)
- Fail-safe: negare per default, consentire dopo una verifica
- Minimizzare l'uso di risorse condivise o non controllate dal programma (mktemp), attenzione alle race condition.
- Chroot, opzioni di configurazione, e altri accessori

Programmazione sicura: how ? (2)

- Attenzione alle chiamate remote (RMI, CORBA, e RPC)
- Attenzione all'uso di librerie: non sapete quanta attenzione gli altri sviluppatori hanno posto nella sicurezza. Parsate anche i risultati delle chiamate (!)
- Attenzione ai dati in formato HTML, XML, SGML
- Attenzione anche all'OUTPUT: potrebbe fornire informazioni a un attaccante, o se rallentato fornire un DOS. Vedremo i problemi delle format string
- Non scrivete codice per gestire le password o algoritmi crittografici, usatene di provati ed affidabili
- Non conservate mai segreti in chiaro, possibilmente nemmeno in memoria !
- Generate i numeri random con sorgenti affidabili d'entropia (/dev/random)

Validare sempre gli input

- Soprattutto: l'input deve essere **sempre** analizzato e validato
- Quale input è inaffidabile ?
  - Input degli utenti, specialmente via rete
    - Non sempre sono chi pensiamo...
  - Input del sistema
    - Es.: file race conditions, variabili d'ambiente (getenv()), vettore dei parametri (argv[i])
- **Come validare l'input ?**
  - Determinare ciò che è AMMESSO e scartare il resto, MAI viceversa.
  - Occorre sempre disattivare i privilegi PRIMA del parsing: un buffer overflow può avvenire ad ogni punto