



Università degli Studi di Salerno

Corso di
Sicurezza su Reti

Anno accademico 2000/01.

StegFS

Studenti:

Caputo Gianluca 056/100210
D'Apice Ezio 056/100613
Inverso Omar 056/100544
Raiola Salvatore 056/100274

Prof.

De Santis Alfredo



File System Steganografico

Contenuto della presentazione:

1. Classificazione degli attacchi
2. File system steganografico
3. Negabilità plausibile
4. Metodi proposti
5. StegFS
6. Utilizzo pratico
7. Conclusioni



Classificazione degli attacchi



Classificazione degli attacchi

- ✓ "Sicurezza di un sistema" spesso vuol dire "capacità di mantenere riservate le informazioni"
- ✓ Messa in sicurezza:
 - Individuazione delle possibili tipologie di attacchi
 - Individuazione ed applicazione della soluzione per ogni possibile attacco



Attacco di I livello

- ✓ L'attaccante è "curioso", non ha accesso fisico al sistema
- ✓ L'attaccante non è disposto a fare minacce o ad utilizzare la forza per violare un sistema (es.: colleghi di lavoro che in ufficio cercano di dare qualche sbirciatina ai documenti altrui...)

Soluzione:

Cifratura dei dati.

Mi basterà cifrare... e conservare la chiave.



Attacco di II livello

- ✓ L'attaccante "determinato" ha accesso fisico al sistema
- ✓ L'attaccante non vuole far rilevare l'attacco (es.: furto di progetti di lavoro, di banche dati importanti...)

Soluzione:

Rendere più sicuro l'edificio che ospita il sistema;

Cifratura più "forte" dei dati (es. utilizzo di chiavi più lunghe).

Mi dispiace ma non posso uscire, mi dispiace ma non posso uscire, mi dispiace ma non posso uscire.





Attacco di III livello

- ✓ L'attaccante, "disposto a tutto", ha accesso fisico al sistema
- ✓ L'attaccante dispone di mezzi di costruzione atti a scoprire la chiave (es.: terrorismo, spionaggio industriale, intrighi internazionali...)

Soluzione:

Utilizzo di file system steganografici a negabilità plausibile

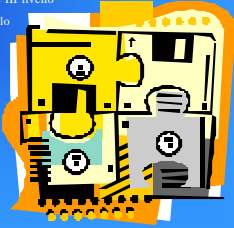


File System Steganografico



File system steganografico

- ✓ Può essere considerato l'estensione naturale di un file system cifrato
- ✓ Assicura protezione contro gli attacchi di I e III livello
- ✓ Protegge parzialmente da attacchi di II livello



File system steganografico in dettaglio

- ✓ I file sono organizzati in insiemi detti *livelli di sicurezza*
- ✓ Ad ogni livello corrisponde una passphrase
- ✓ Non conoscere la passphrase di un livello di sicurezza equivale ad ignorare completamente qualsiasi informazione sul livello e sul file in esso contenuti

(Differenza fondamentale fra file system cifrati e file system steganografati)



Ancora sui livelli di sicurezza

- ✓ La gerarchia più utilizzata, detta "ad accesso lineare" prevede che l'accesso al livello i dia accesso anche ai livelli da $i-1$ a 1.
- ✓ Chi attacca sa quanti sono i livelli di sicurezza, dato che questo è un numero fissato
- ✓ Chi attacca non saprà mai quanti livelli sono effettivamente utilizzati (e quindi quante passphrase deve scoprire)

(In questo caso si dice soddisfatta la proprietà di *negabilità plausibile* sul numero di livelli utilizzati)



Negabilità plausibile

✓ Def.:

Sia dato un meccanismo di sicurezza con parametri $\{p_1, p_2, \dots, p_n\}$. Tale meccanismo gode della proprietà di *negabilità plausibile* su p_j se e solo se è possibile mentire in maniera del tutto plausibile sul valore di p_j .

- ✓ Il meccanismo di sicurezza è il file system steganografico
- ✓ Il parametro p_j è il numero di livelli di sicurezza effettivamente utilizzati





Negabilità plausibile: esempio

- ✓ Si supponga di avere un file system steganografico con 7 livelli di sicurezza
- ✓ Si supponga che l'utente utilizzi solo 5 livelli
- ✓ Il numero effettivo di livelli utilizzati non è memorizzato su disco né ricavabile
- ✓ L'utente, se minacciato, può falsamente confessare di aver utilizzato solo i primi 3 livelli
- ✓ L'attaccante non può in nessun modo verificare se i livelli di sicurezza effettivamente utilizzati sono davvero 3
- ✓ L'attaccante dovrà rassegnarsi, accontentandosi di avere accesso ai soli file dei primi tre livelli



Metodi proposti per la steganografia di file system



Metodi proposti

I metodi proposti, finora, per l'implementazione di file system steganografici, sono tre:

- ✓ Primo metodo: proposto da *Anderson, Needham e Shamir* nel 1998
- ✓ Secondo metodo: proposto da *Anderson, Needham e Shamir* nel 1998
- ✓ Terzo metodo: proposto da *Van Schaik e Smeddle*



Primo metodo

- ✓ Proposto da *Anderson, Needham e Shamir*
- ✓ Lo schema opera su un set di file "mascherati" con inizialmente un contenuto casuale
- ✓ La memorizzazione consiste nel modificare i file mascherati
- ✓ I file mascherati devono essere XORati insieme in modo da ricostruire il file nascosto
- ✓ Si assume che chi attacca non ha conoscenza del testo in chiaro memorizzato
- ✓ Sono richieste solo operazioni di algebra lineare e non forti cifrari a blocchi
- ✓ La password di accesso al file identifica un sottoinsieme dei file mascherati

I principali svantaggi di questo metodo sono

- ✓ Scarse prestazioni (dovute all'elevato numero di file mascherati)
- ✓ Assunzione del fatto che l'attaccante ignora qualsiasi parte del testo in chiaro (dovuta alla necessità di utilizzare algebra lineare al posto di cifrari più forti)
- ✓ Manca il supporto per i file di lunghezza qualsiasi e per le directory



Secondo metodo

- ✓ Proposto da *Anderson, Needham e Shamir*
- ✓ Il file system viene all'inizio completamente riempito da blocchi di dati casuali
- ✓ I blocchi dei file cifrati sono nascosti fra i blocchi casuali scrivendoli in locazioni pseudo-casuali (viene utilizzata una chiave derivata dall'hash del nome/persorso del file)
- ✓ **I blocchi dei file cifrati sono indistinguibili da quelli riempiti in modo casuale**
- ✓ Per limitare le sovrascritture (e quindi la perdita dei dati) ogni blocco viene duplicato m volte

I principali svantaggi di questo metodo sono:

- ✓ Possibile perdita di dati (dovute alle possibili sovrascritture dei blocchi di dati)
- ✓ Leggero overhead in lettura/scrittura (dovuto alla necessità di duplicare i blocchi)



Terzo metodo

- ✓ Proposto da *Van Schaik e Smeddle*
- ✓ Nel file system viene nascosta l'informazione effettuando una combinazione lineare di sottoinsiemi di blocchi
- ✓ Vengono marcati i blocchi di dati con qualcosa del tipo "potrebbe essere usato nei livelli superiori di sicurezza"

I principali svantaggi di questo metodo sono:

- ✓ Mancanza di una effettiva negabilità plausibile sulla presenza di altri dati cifrati (dovuta alla marcatura dei blocchi nascosti)
- ✓ Possibilità di ricavare un limite superiore stretto sulla quantità di blocchi nascosti
- ✓ Possibilità di sapere, partendo da un dato livello di sicurezza, se ne esistono altri





Il file system steganografico di Linux: StegFS



Introduzione a StegFS

- ✓ StegFS è una libera implementazione di file system steganografico su piattaforme Linux
- ✓ Descritto nel 1999 da *Andrew D. McDonald* e *Markus G. Kuhn*
- ✓ Implementato da *Andrew D. McDonald*
- ✓ Distribuito sotto licenza GNU GPL (General Public License)
- ✓ Ispirato al secondo metodo proposto da *Anderson, Needham* e *Shamir*



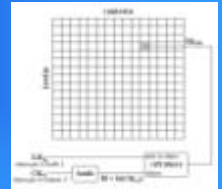
Gestione delle chiavi

- ✓ Ogni file nascosto appartiene ad uno dei 15 livelli di sicurezza
- ✓ E' consentito l'accesso simultaneo a gruppi di livelli, tramite i contesti di sicurezza
- ✓ Un contesto di sicurezza è identificabile come un insieme (contenitore) di livelli
- ✓ Per default, il contesto di sicurezza *C* dà accesso a tutti i livelli da 1 a *C*
- ✓ Il numero del livello è sempre relativo al contesto, quindi per accedere ad un livello è necessario attivare il contesto che lo contiene
- ✓ E' possibile costituire gerarchie più complesse fra livelli e contesti, evitando quindi la tradizionale gerarchia ad accesso lineare



Gestione delle chiavi

- ✓ La gestione delle chiavi è organizzata tramite una matrice 15x15, detta matrice di sicurezza
- ✓ Le righe identificano i livelli e le colonne i contesti
- ✓ Ogni elemento contiene la chiave di livello cifrata usando come chiave l'hash della chiave di contesto
- ✓ All'apertura di un contesto di sicurezza, StegFS utilizza l'hash della chiave di contesto fornita dall'utente per decifrare l'intera colonna delle chiavi di livello appartenenti a quel contesto
- ✓ Grazie alle chiavi di livello ottenute, StegFS è in grado di cifrare e decifrare i blocchi dei file appartenenti a tali livelli



Allocazione dei blocchi

- ✓ Gli inode vengono posizionati in blocchi a posizione casuale, effettuando l'hashing dell'indice dell'inode più la chiave per il livello di sicurezza
- ✓ I blocchi di dati sono posizionati in locazioni completamente casuali (/dev/urandom genera numeri casuali)
- ✓ Viene mantenuta una tabella di allocazione dei blocchi per tener traccia dei blocchi allocati e di altri dati relativi ai singoli blocchi (un'entrata per ogni blocco su disco)



Cifratura dei blocchi su disco

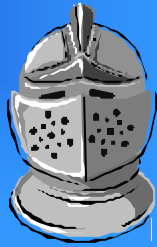
- ✓ Ogni blocco è separatamente cifrato nella modalità CBC (Cipher Block Chaining)
- ✓ Viene utilizzato un vettore di inizializzazione (IV) memorizzato all'interno della corrispondente entrata nella tabella di allocazione dei blocchi
- ✓ Se il blocco *i* appartiene ad un file nascosto sotto il livello *L*, allora il file e la relativa entrata nella tabella dei blocchi sono cifrati dal valore ottenuto effettuando lo XOR fra la chiave del livello di sicurezza e il numero di blocco





Implementazione di StegFS

- ✓ Viene installato in aggiunta ai normali driver di file system (Ext2fs, Minix, ...)
- ✓ Il driver StegFS può lavorare su partizioni Ext2fs e viceversa
- ✓ Segue da vicino la semantica di un file system standard per un sistema Unix (ci sono sottodirectory, inode, link simbolici, hard link, ...)



Approccio utilizzato per l'implementazione di StegFS

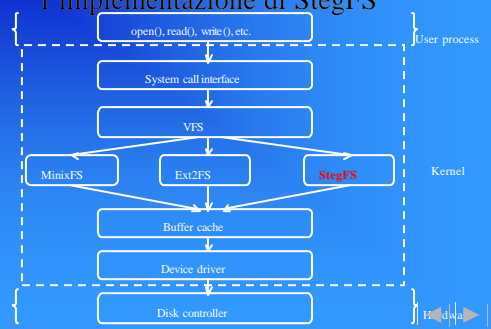


Tabella di allocazione dei blocchi

- ✓ Sostituisce il bitmap di allocazione dei blocchi utilizzato per Ext2fs
- ✓ In Ext2fs (bitmap) c'è un bit per blocco, in StegFS ci sono 128 bit per blocco
- ✓ Lo scopo principale è memorizzare la checksum cifrata per ogni blocco, in modo tale da rilevare i blocchi sovrascritti
- ✓ E' memorizzata in un file separato, non nascosto
- ✓ Ogni entrata viene cifrata con la stessa chiave utilizzata per la cifratura dei dati per i corrispondenti blocchi del disco



Tabella di allocazione dei blocchi

- ✓ Consiste nella concatenazione di tre variabili di 32 bit e due di 16 bit:


```

struct stegfs_btable {
    uint32_t magic1;
    uint16_t magic2;
    uint16_t iv;
    uint32_t checksum;
    uint32_t ino;
};
      
```
- ✓ **magic1** dovrebbe essere sempre nulla
- ✓ **magic2** è 1 se il corrispondente blocco contiene un inode, 0 per un blocco di dati
- ✓ **magic1** e **magic2** insieme contengono 47 bit di ridondanza che permettono di determinare se un blocco è usato ad un livello di sicurezza per il quale si è a conoscenza della chiave
- ✓ **iv** è il vettore di inizializzazione per la cifratura
- ✓ **checksum** contiene gli ultimi 32 bit del contenuto del blocco cifrato (utilizzati come checksum)
- ✓ **ino** contiene il numero di inode di questo file



Allocazione dei blocchi (algoritmo)

ALGORITMO DI ALLOCAZIONE DEI BLOCCHI

- 1 Prima di allocare un blocco, viene controllato il bitmap Ext2fs
- 2 Se il blocco è libero in Ext2fs, allora per ogni chiave di livello conosciuta viene decifrata l'entrata corrispondente nella tabella dei blocchi
- 3 Se i primi 47 bit (**magic1** + **magic2**) non corrispondono a 0, il blocco viene allocato, altrimenti si passa al blocco successivo



VFS

- ✓ Inode generici e strutture di superblocco in memoria
- ✓ Campi addizionali nel superblocco per StegFS (rispetto ad Ext2fs):
 - campi per gestire i livelli di sicurezza (es. chiavi per il livello correntemente aperto)
 - puntatore alle funzioni di cifratura
 - puntatore della tabella dei blocchi



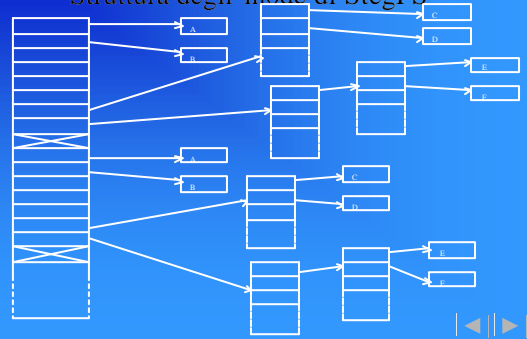


Inode e blocchi

- ✓ Gli inode di StegFS sono quasi identici a quelli di Ext2fs
- ✓ Come in Ext2fs, gli inode di StegFS sono composti da 12 blocchi diretti, 1 indiretto, 1 doppiamente indiretto ed uno triplamente indiretto
- ✓ Addizionalmente ci sono più copie per ogni inode e per ogni blocco di dati
- ✓ La grandezza di un inode di StegFS è 1024 byte
- ✓ Si possono avere fino a 14 copie dello stesso blocco di dati e 28 per ogni inode



Struttura degli inode di StegFS



Indice degli inode

- ✓ E' un intero a 32 bit che identifica univocamente un file all'interno del file system
- ✓ In Ext2fs la posizione dell'inode di un file su disco può essere calcolata direttamente a partire dall'indice del suo inode
- ✓ In StegFS è necessario ricercare l'inode all'interno della tavola dei blocchi decifrata, e sperare di trovarne uno che non sia stato sovrascritto
- ✓ I file nascosti vengono distinti da quelli normali in base all'indice dell'inode



Indice per un inode tradizionale(Ext2fs)



Indice per un inode nascosto (StegFS)



Replicazione dei blocchi

- ✓ I blocchi usati da StegFS sono contrassegnati come non allocati in Ext2fs, quindi possono essere sovrascritti
- ✓ E' possibile fare più copie per blocco(inode) (parametro configurabile)
- ✓ E' possibile verificare in lettura l'esistenza di un blocco non sovrascritto tramite la checksum



Compilazione ed installazione (modulo)

- ✓ Compilazione ed installazione per un kernel versione 2.2.18
- ✓ Posizionarsi in `/usr/src/linux`
- ✓ Applicare la crypto patch
`patch -p1 < ./patch_int_2.2.18.3`
- ✓ Posizionarsi nella radice dei sorgenti di StegFS
- ✓ Compilare come modulo
`make`
- ✓ Installare StegFS
`make install`



Per una corretta installazione del modulo impartire il comando `depmod -a`



Compilazione ed installazione (patch)

- ✓ Posizionarsi nella radice dei sorgenti di StegFS
- ✓ Generare la patch:
`make patch`
- ✓ Applicare la patch:
`make applypatch`
- ✓ Compilazione dei comandi:
`make tools`
- ✓ Installazione dei comandi:
`make installtools`



Poi è necessario compilare il kernel di Linux con StegFS





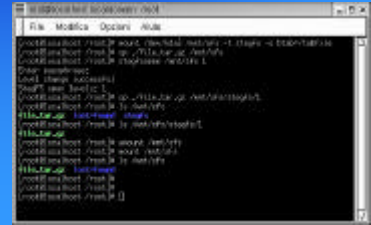
Utilizzo pratico di StegFS

- ✓ Creare una partizione con qualche tool (fdisk, diskdruid, etc...)
- ✓ Creare un file system di tipo Ext2FS:
`mke2fs /dev/blockdevice`
- ✓ Creare un file system di tipo StegFS:
`mkstegfs /dev/blockdevice /path/to/hiab_file`



Utilizzo pratico di StegFS

- ✓ Montare una partizione di tipo StegFS:
`mount /dev/blkdev /mnt/point -t stegfs -o btab=/path/to/btab`
- ✓ Apertura finalizzata al deposito o all'accesso a directory e file steganografati:
`stegfsopen /mnt/mntpoint contextnum`



Comandi di StegFS (mkstegfs)

- ✓ Per generare un file system steganografico StegFS a partire da una partizione Ext2FS:
`mkstegfs /dev/fsdevice btabfile [-n numlevels] [-c cipher] [-i inocopies] [-b blkcopies]`

Descrizione delle opzioni:	
<i>fsdevice</i>	Partizione StegFS Ext2fs
<i>btabfile</i>	File che conterrà la tabella dei blocchi dei file memorizzati
<i>numlevels</i>	Numero di livelli di sicurezza da definire. E' per default 15 (tutti i livelli). Se si specifica un valore diverso, vuol dire che l'utente non desidera impostare le passphrase per ogni livello.
<i>cipher</i>	Nome dell'algoritmo di cifratura
<i>inocopies</i>	Numero di copie da effettuare per ogni inode (5 per default)
<i>blkcopies</i>	Numero di copie da effettuare per ogni blocco di dati (5 per default)



Comandi di StegFS (stegfsopen)

- ✓ Apre tutti i livelli di un contesto di sicurezza, o il livello specificato all'interno del contesto:
`stegfsopen /mnt/mntpoint [contextnum [levelnum]]`

- ✓ Descrizione delle opzioni:

<i>mntpoint</i>	Punto di mount del file system steganografato
<i>contextnum</i>	Indice del contesto di sicurezza da aprire
<i>levelnum</i>	Indice del livello di sicurezza appartenente al contesto specificato



Comandi di StegFS (rerpl, stegfsclose)

- ✓ Rigenera i blocchi persi in un file steganografato:

Descrizione delle opzioni:	
<i>file</i>	Specifica il file da rigenerare.

- ✓ Chiude un livello di sicurezza:

`stegfsclose /mnt/mntpoint [levelnum]`

- ✓ Descrizione delle opzioni:

<i>mntpoint</i>	Punto di mount del file system steganografato
<i>levelnum</i>	Livello di sicurezza da chiudere. Se non specificato saranno chiusi tutti i livelli



Comandi di StegFS (stegfscrtl, tunestegfs)

- ✓ Altera i contesti di sicurezza:

`stegfscrtl [options] /dev/fsdevice btabfile [cmd] [cmdoptions]`

- ✓ Descrizione delle opzioni:

<i>fsdevice</i>	Device associato al file system
<i>btabfile</i>	File in cui è memorizzata la tabella di allocazione dei blocchi per i file steganografati
<i>cmd</i>	Può essere <i>add</i> , <i>remove</i> o <i>init</i>

- ✓ Controlla o imposta il numero di copie per blocco di dati e per inode per i file steganografati
`tunestegfs [-i inocopies] [-b blkcopies] file`

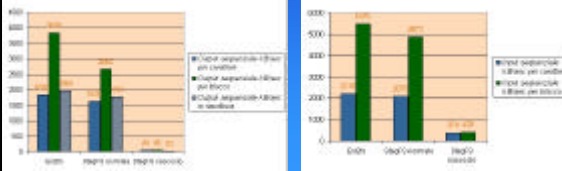
- ✓ Descrizione delle opzioni:

<i>inocopies</i>	Regola il numero di copie per ogni inode
<i>blkcopies</i>	Regola il numero di copie per ogni blocco di dati
<i>file</i>	File di cui si vogliono regolare i parametri



Prestazioni

- ✓ Il test è stato effettuato dagli autori di StegFS su un processore AMD K5 PR150 100Mhz, utilizzando una partizione di 1 Gb su disco IDE Fujitsu da 1.2Gb
- ✓ Sono confrontate le prestazioni di Ext2FS con quelle di StegFS utilizzando un fattore di replicazione pari a 5, sia per gli inode che per i blocchi di dati dei file nascosti



Conclusioni



- ✓ Negabilità plausibile sul numero di file memorizzati su disco
- ✓ Riservatezza del contenuto di tutti i file nascosti
- ✓ Distruzione sicura dei file nascosti
- ✓ Possibilità di utilizzare diversi strati di accessi a negabilità plausibile, in modo da compromettere volontariamente gli strati inferiori, senza rivelare quelli superiori
- ✓ Negabilità plausibile su strati di sicurezza superiori a quello eventualmente scoperto
- ✓ Giustificazione plausibile sulla presenza del driver, rivelando solo lo strato più basso e utilizzando i vantaggi della sicurezza aggiuntiva offerti dal prodotto
- ✓ Moderato numero di accessi in scrittura mentre non tutti gli strati nascosti sono aperti, in modo da non danneggiare i dati appartenenti a file nascosti
- ✓ Indistinguibilità degli accessi in scrittura ai file steganografati rispetto ai file in chiaro creati e cancellati
- ✓ Piena compatibilità con Ext2FS



Autori

Caputo Gianluca caputo@interfree.it
 D'Apice Ezio edapice@inwind.it
 Inverso Omar omainv@interfree.it
 Raiola Salvatore raiola@inwind.it

