Protezione del Kernel Tramite Macchine Virtuali

Fabio Campisi
Corso di Laurea d'Informatica, Università di Pisa
Pisa, Italy
campisi@cli.di.unipi.it

Daniele Sgandurra
Dipartimento di Informatica, Università di Pisa
Pisa, Italy
daniele@di.unipi.it

Abstract

In questo paper proponiamo una metodologia per proteggere l'integrità del kernel di un sistema operativo tramite l'utilizzo delle macchine virtuali (MV). L'approccio adottato sfrutta la capacità offerta dal gestore delle macchine virtuali di accedere direttamente dal livello hardware/firmware ad ogni componente allocata alle MV. Questo permette, tramite la tecnica dell'introspezione delle macchine virtuali, di effettuare controlli di consistenza sulle strutture dati e sul codice del kernel da un livello inferiore a quello del kernel. Inoltre, descriviamo l'implementazione di una libreria di funzioni per effettuare un insieme di controlli di consistenza seguendo l'approccio proposto. I controlli hanno come oggetto il kernel Linux e utilizzano, come tecnologia di virtualizzazione, l'hypervisor Xen. L'overhead, introdotto dall'applicazione periodica dei controlli utilizzando la libreria proposta, è inferiore al 10% nel caso peggiore.

Keywords

Introspezione, macchine virtuali, integrità del kernel

1 INTRODUZIONE

Negli ultimi anni si è verificato un notevole aumento nel grado di sofisticazione degli attacchi verso i sistemi informatici. Ad esempio, i *rootkits*, gli strumenti software utilizzati dagli attaccanti per celare la loro presenza e per mantenere il controllo del sistema compromesso, si sono evoluti da quelli eseguiti a livello utente a quelli a livello kernel [14, 35]. Questo dimostra che gli attaccanti cercano di occupare il livello più basso del sistema per: (i) ottenere il controllo totale del sistema; (ii) ingannare il legittimo amministratore del sistema nascondendo le tracce della compromissione e fornendo informazioni false sullo stato del sistema, così da infondere un falso senso di fiducia all' amministratore. Dato che sia gli strumenti di attacco, i rootkits, che gli strumenti di difesa, ad esempio i sistemi per il rilevamento delle intrusioni (Intrusion Detection System, IDS), possono essere eseguiti a livello kernel, siamo in presenza di una continua lotta a chi per primo riesce a predire e prevenire le mosse dell'avversario per riuscire ad ottenere il controllo totale del sistema [34].

In questo contesto, una tecnica generica e indipendente dal livello architetturale per rilevare intrusioni ad un sistema informatico è quella della *introspezione*. Questa tecnica permette di analizzare lo stato di un sistema per ricostruire le strutture dati critiche utilizzate all'interno del sistema stesso e controllarne la loro consistenza. Nel caso in cui si intenda adottare questa tecnica a livello hardware/firmware e non si disponga di un supporto specifico, come la presenza di un coprocessore dedicato ai controlli di integrità [32], è possibile utilizzare la tecnica della *introspezione delle macchine virtuali* (Virtual Machine Introspection, VMI) [18] che richiede di eseguire i sistemi operativi da monitorare all'interno di macchine virtuali (MV). Una MV è un ambiente di esecuzione, creato da una tecnologia di virtualizzazione, che emula a software il comportamento dell'architettura hardware/firmware sottostante [9, 38]. Il gestore delle macchine virtuali (Virtual Machine Monitor, VMM) [17] è un piccolo programma eseguito direttamente sopra il livello hardware/firmware per creare, gestire e monitorare un insieme di MV. Grazie a questa tecnologia, un computer standard è in grado di eseguire in concorrenza un insieme di MV, ognuna delle quali esegue un diverso sistema operativo. Inoltre, sfruttando l'accesso diretto del VMM a tutte le componenti allocate ad ogni macchina virtuale, la tecnica VMI permette di analizzare lo stato del kernel in esecuzione all'interno delle MV. In questo modo, dato che la tecnica VMI permette di controllare la consistenza del kernel da un livello inferiore rispetto a quello che un attaccante può ottenere, cioè il livello kernel, diventa molto difficile attaccare direttamente o evadere i controlli effettuati tramite l'utilizzo della tecnica VMI. In quest'ottica, abbiamo progettato un insieme di controlli per proteggere l'integrità del kernel di un sistema operativo tramite VMI. Nel seguito indicheremo con *Mon-*

MV una macchina virtuale su cui è in esecuzione il kernel che vogliamo monitorare. Questo kernel viene protetto tramite VMI da un'altra macchina virtuale, che chiameremo *I-MV*, cioè la macchina virtuale di introspezione. Inoltre, indicheremo con *introspettore* il modulo delegato ai controlli di sicurezza in esecuzione sulla I-MV che ha il compito di rilevare gli attacchi contro il kernel in esecuzione su una Mon-MV. Questo è reso possibile perché il VMM, il gestore delle MV, esporta una *interfaccia di controllo* per permettere ad una MV privilegiata di gestire le altre MV. Questa interfaccia può essere anche utilizzata per accedere direttamente allo stato di ogni Mon-MV in esecuzione e monitorare l'integrità del kernel applicando la tecnica VMI.

Il resto del paper è organizzato come segue. La Sez. 2 discute l'approccio generale seguito e introduce *Xen-VMI*, una libreria di introspezione per applicare un insieme di controlli di consistenza su un kernel Linux in esecuzione su Xen. La Sez. 3 descrive l'implementazione di Xen-VMI. La Sez. 4 presenta una valutazione dei risultati di sicurezza del prototipo realizzato e discute anche l'overhead introdotto da questa soluzione e ne mette in evidenza gli attuali limiti. La Sez. 5 presenta una rassegna di lavori simili. Infine, nella Sez. 6 presentiamo le nostre conclusioni e delineiamo alcune idee per sviluppi futuri.

2 APPROCCIO GENERALE

Per poter effettuare i controlli di protezione del kernel, il nostro approccio richiede che i sistemi operativi che devono essere monitorati siano eseguiti all'interno di macchine virtuali, chiamate Mon-MV. Queste MV sono eseguite concorrentemente su uno stesso computer su cui sia presente un VMM che supporta la creazione, gestione e arresto delle MV. Inoltre, un'altra MV, chiamata I-MV, è dedicata all'applicazione periodica dei controlli di integrità tramite la tecnica VMI sul kernel in esecuzione sulle Mon-MV. La I-MV accede all'interfaccia di controllo esportata dal VMM e verifica a tempo di esecuzione un insieme di invarianti che devono essere valide per il kernel. Infatti, l'I-MV analizza la memoria allocata ad ogni Mon-MV per rilevare possibili violazioni alla sicurezza. Inoltre, se un attacco viene sospettato, la I-MV gestisce l'evento tramite l'applicazione di un'azione, come ad esempio bloccare l'esecuzione della relativa MV. La capacità di "congelare" l'esecuzione di una MV è un altro vantaggio e peculiarità delle tecnologie di virtualizzazione. Questo permette alla I-MV, tra le altre cose, di esaminare in dettaglio lo stato di una MV quando questa è bloccata.

Inoltre, sottolineiamo che il punto di partenza di questo approccio, salvaguardare l'integrità del kernel dal livello VMM, può costituire il primo passo per realizzare una *chain of trust*. Ad esempio, è possibile estendere il kernel in esecuzione all'interno di una Mon-MV per utilizzare funzionalità aggiuntive di sicurezza già esistenti, come quelle offerte da SELinux [28, 29]. In questo contesto, riteniamo che la robustezza totale del sistema monitorato sia elevata in quanto, oltre al kernel, è possibile proteggere in maniera più affidabile anche i processi a livello utente. Questo approccio garantisce così la sicurezza del sistema in due passi:

1) tramite I-MV si assicura l'integrità del kernel in esecuzione all'interno di una Mon-MV;
2) il kernel, esteso con ulteriori funzioni di sicurezza, controlla la consistenza e l'integrità dei processi a livello utente.

Tramite la tecnica VMI si fornisce assicurazione che le componenti critiche in esecuzione all'interno di una MV, come il kernel, non siano manomesse e, quindi, si garantisce che il *trusted computing base* (TCB) sia esteso per includere tutte le componenti critiche protette tramite introspezione. Un'assunzione importante in questo contesto è che il VMM sia affidabile e che l'introspezione sia una tecnica sicura per estendere il TCB. La ragione che ci porta a ritenere che il VMM e la VMI siano affidabili è duplice. In primo luogo, il VMM ha piena visibilità di ogni Mon-MV poiché accede ad esse dal livello hardware/firmware, ed è quindi più difficile eluderne i controlli. In secondo luogo, il VMM è più robusto di un sistema operativo tradizionale, come Linux o Windows, perché: (i) l'interfaccia che esporta ai livelli superiori è molto limitata rispetto a quella esportata da un sistema operativo che, ad esempio, implementa qualche centinaio di chiamate di sistema, ed è quindi più difficile sfruttare le vulnerabilità di un VMM; (ii) la dimensione ridotta del codice di un VMM riduce il numero di opportunità che un attaccante ha per comprometterlo, e ne rende anche più facile validarne la sua corretta implementazione tramite strumenti formali di analisi statica del codice. In conclusione, dato che il VMM ha sempre piena visibilità delle MV in esecuzione ed è isolato dalle MV che gestisce, la complessità di compromettere un VMM o di eluderne i controlli effettuati tramite introspezione è molto elevata. Nondimeno, ci sono minacce contro il VMM che devono essere comunque prese in considerazione [11].

2.1 Macchina Virtuale d'Introspezione

La macchina virtuale di introspezione (I-MV) esegue un modulo delegato all'applicazione periodica dei controlli, chiamato introspettore. Questo modulo monitora le componenti critiche del kernel in esecuzione su una Mon-MV per rilevare attacchi. Nel dettaglio, l'introspettore monitora la sezione *text* del kernel e dei moduli, cioè quella sezione contenente il codice del kernel, per verificare se sia stata modificata. Infatti, le regioni della memoria principale contenenti queste sezioni sono marcate dal sistema operativo in sola lettura e, quindi, ogni tentativo di modificarle suggerisce che un attaccante stia tentando di inserire ed eseguire istruzioni illecite. Ad esempio, questo avviene quando un rootkit di livello kernel, tramite l'inserimento nel kernel di un modulo [20], cerca di modificare il codice di una chiamata di sistema o del gestore delle chiamate di sistema. Inoltre, dato che un rootkit può anche modificare puntatori all'interno di strutture dati critiche, come la tabella dei puntatori per le chiamate di sistema [16, 25], l'introspettore controlla anche l'integrità delle strutture dati critiche per il kernel. Infine l'introspettore, nel momento in cui verifica che il kernel in esecuzione su una Mon-MV è stato modificato, gestisce l'evento corrispondente tramite l'esecuzione di un'azione, tra cui: (i) blocca l'esecuzione della MV sotto attacco e salva il suo stato su un file; (ii) chiude la connessione della MV alla rete locale; (iii) termina i processi eseguiti dall'attaccante; (iv) disconnette l'attaccante dalla MV; (v) invia un alert alla console di amministrazione con informazioni relative alla possibile compromissione.

L'introspettore utilizza una libreria, che abbiamo chiamato *Xen-VMI,* che rende possibile ricostruire una visione a livello di sistema operativo dello stato di esecuzione di una MV a partire dai dati "grezzi" ottenuti tramite la tecnica VMI. Questa liberia ha un ruolo fondamentale perché l'interfaccia di controllo esportata dal VMM offre solamente una visione di basso livello dello stato di una Mon-MV, cioè una visione a livello di registri del processore, blocchi del disco e array di memoria. Tuttavia, l'introspettore per effettuare i controlli di consistenza deve poter ragionare con concetti tipici di un sistema operativo, quali quelli di file, processo e memoria virtuale. Quindi, la libreria Xen-VMI ha il compito di tradurre la visione di basso livello ottenuta tramite VMM in una di più alto livello in termini di strutture dati utilizzate dal kernel. Per fare questo, la libreria Xen-VMI deve conoscere esattamente il kernel in esecuzione su ogni Mon-MV, cioè l'implementazione delle strutture dati e dove queste sono allocate in memoria. In questo modo, la libreria Xen-VMI mette a disposizione dell'introspettore una visione a livello di sistema operativo dello stato di una MV in esecuzione per poter ottenere informazioni generali come la lista dei processi in esecuzione, la lista dei moduli caricati nel kernel o anche informazioni relative ad un identificatore di processo, coma la lista dei file o socket aperti. Questo tipo di informazione può essere utilizzato per implementare un controllo di consistenza generico e molto efficace: l'introspettore, tramite la libreria Xen-VMI, ottiene una lista di strutture dati del kernel e confronta questa lista con quella ottenuta eseguendo un comando di utilità all'interno del sistema operativo monitorato. Qualsiasi differenza tra questi due risultati può significare che un attaccante ha sostituito binari critici del sistema operativo per nascondere le proprie attività.

2.2 Macchina Virtuale Monitorata

La Mon-MV esegue il sistema operativo che vogliamo monitorare. Come già accennato precedentemente, il kernel in esecuzione in questa MV può essere esteso per includere ulteriori controlli di sicurezza, come Grsecurity [19] o Lids [26], che aumentano sia la robustezza del sistema operativo stesso che quella dei processi a livello utente. Ad esempio, Grsecurity è un insieme di patch per il kernel di Linux che permette, tra l'altro, di prevenire attacchi che sfruttano *buffer overflow,* come l'esecuzione di codice arbitrario nel kernel. Invece, LIDS (Linux Intrusion Detection System) è un insieme di patch per il kernel con lo scopo di proteggere il file system da manomissioni (ad esempio, al file /bin/login), rilevare i port e gli stealth scan, ed implementare politiche MAC (Mandatory Access Control).

3 IMPLEMENTAZIONE DELLA LIBRERIA XEN-VMI

La libreria Xen-VMI è stata realizzata utilizzando il linguaggio C. Il VMM che abbiamo utilizzato è Xen versione 3.0.2, che è un hypervisor open source realizzato all'Università di Cambridge. Il sistema operativo in esecuzione sulle Mon-MV è Debian-Linux, la cui versione del kernel è la 2.6.16. Abbiamo utilizzato anche la libreria XenAccess [40] come base per realizzare Xen-VMI, in particolare per ottenere gl'indirizzi virtuali delle strutture dati critiche del kernel in esecuzione su una Mon-MV e mappare le relative pagine all'interno dello spazio di indirizzamento del introspettore. Infine, abbiamo utilizzato la libre-

ria OpenSSL [30] per eseguire alcune funzioni accessorie, come calcolare gli hash di regioni critiche di memoria del kernel. Il codice sorgente della libreria Xen-VMI è disponibile all'indirizzo:
http://www.di.unipi.it/~sgandurra/projects/projects.php.

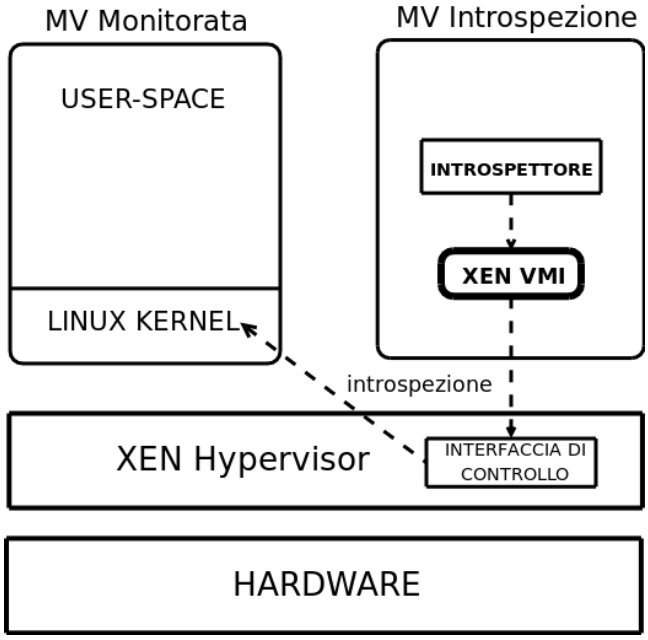

Figura 1: Architettura

3.1 Funzioni di Introspezione

Nei paragrafi seguenti analizziamo l'approccio seguito da tutte le funzioni di introspezione e mostriamo l'implementazione di alcune di esse per illustrare i principi generali della libreria Xen-VMI.

Strategia Generale per Effettuare i Controlli

Per poter realizzare in maniera corretta le funzioni di introspezione, abbiamo utilizzato gli header del kernel di Linux in esecuzione all'interno delle Mon-MV. Questo ha permesso alla libreria Xen-VMI di ricostruire esattamente le strutture dati utilizzate dal kernel Linux a partire dai dati di basso livello ottenuti tramite l'interfaccia di controllo del VMM. I passi fondamentali seguiti da queste funzioni per ricostruire le strutture dati ed effettuare i controlli di consistenza sono essenzialmente sei, e sono elencati di seguito:

1. Si congela l'esecuzione della Mon-MV.
2. Si mappano le pagine di memoria di una Mon-MV contenenti le strutture dati critiche del kernel che si vogliono monitorare all'interno dello spazio di indirizzamento dell'introspettore. In questo passo si utilizza il file System.map, relativo al kernel in esecuzione sulla Mon-MV, per ottenere l'indirizzo virtuale associato al simbolo relativo alla struttura dati ricercata.
3. Dopo aver mappato le pagine nello spazio di indirizzamento dell'introspettore, la libreria Xen-VMI effettua il “casting” della memoria mappata per ottenere le strutture dati corrispondenti a quelle utilizzate dal kernel. Come già detto in precedenza, questo è reso possibile in quanto la dichiarazione delle strutture dati è estratta dagli header del kernel di Linux.
4. Se all'interno della struttura dati è presente un puntatore, ad esempio se la struttura è una lista linkata, Xen-VMI mappa la pagina contenente l'indirizzo virtuale puntato ed effettua il casting alla struttura dati opportuna come descritto al passo 3.
5. L'introspettore effettua i controlli di consistenza sulla struttura dati ricostruita. Se ci sono ancora altri puntatori da analizzare, ritorna al punto 3.
6. Se non ci sono violazioni all'integrità delle strutture dati controllate, l'introspettore riavvia l'esecuzione della Mon-MV.

Per illustrare meglio questi passi, nel seguito viene mostrata, come esempio, l'implementazione della funzione di Xen-VMI per ricostruire tramite introspezione la lista dei processi in esecuzione su una Mon-MV. In neretto sono messi in evidenza i punti precedentemente descritti.

```
int get_process_list(uint32_t dom)
{
  xa_instance_t xai;
  unsigned char *memory = NULL;
  struct task_struct *task;
  uint32_t init_addr,next_addr,offset;
  char *symbol = "init_task";

  if(xa_init(dom, &xai) == XA_FAILURE)
    return error_exit(&xai);
1.xc_domain_pause(xai.xc_handle,dom);
```

```
2.memory=xa_access_kernel_symbol(
        &xai, symbol, &offset);
  if(memory == NULL)
    return error_exit(&xai);
  if(linux_map_symbol_to_addr(
    &xai,symbol,&init_addr)==
    XA_FAILURE)
    return error_exit(&xai);

  while(next_addr != init_addr)
  {
3.task=(struct task_struct *)(memory
       + offset);
  next_addr=(uint32_t)next_task(task);
  munmap(memory, XA_PAGE_SIZE);
4.memory=xa_access_virtual_address(
        &xai, next_addr, &offset);
5./*do something with task*/
  }
6.xc_domain_unpause(xai.xc_handle,dom);
  xa_destroy(&xai);
  if(memory)
    munmap(memory, XA_PAGE_SIZE);
  return 0;
}
```

Funzione per Rilevare Modifiche al Kernel

Questa funzione di introspezione rileva attacchi sia al codice che a strutture critiche del kernel. L'introspettore controlla le pagine di memoria della Mon-MV contenenti:

1) il codice del kernel, contenuto tra gli indirizzi riferiti dai simboli *_text* e *_etext*;
2) la tabella dei puntatori per le chiamate di sistema, contenuta nell'array riferito dal simbolo *sys_call_table*;
3) la tabella dei descrittori delle interruzioni, contenuta nella tabella riferita dal simbolo *idt_table*. In realtà, poiché Xen utilizza la tecnica della paravirtualizzazione, sono necessarie alcune modifiche al kernel del sistema operativo per poter essere eseguito all'interno delle MV. Una di queste modifiche che Xen effettua è relativa proprio alla tabella delle interruzioni, che viene sostituita da una tabella gestita da Xen chiamata *trap_table* [37], per cui i controlli di consistenza sono effettuati su questa tabella invece che sulla *idt_table*.

Poiché queste pagine non devono essere modificate, l'introspettore periodicamente calcola l'hash di esse e verifica che non siano state aggiornate. Per rilevare modifiche illegali a pagine che invece sono marcate in scrittura, un insieme di invarianti deve essere precalcolato per poter poi essere valutato a tempo di esecuzione [13]. Nella Sez.6 approfondiremo questo aspetto.

Controlli sui Loadable Kernel Module

Linux supporta il concetto di *loadable kernel module* (LKM) per poter estendere dinamicamente un kernel in esecuzione, inserendo nuove funzionalità solo quando queste sono effettivamente richieste. Inoltre, quando una funzionalità fornita da un modulo non è più necessaria, un LKM può essere facilmente rimosso dal kernel. Sebbene questa modalità per estendere il kernel in esecuzione sia conveniente ed efficiente, nondimeno introduce serie minacce per la sicurezza del sistema. Ad esempio, un attaccante, dopo aver compromesso un sistema, può utilizzare questa metodologia per inserire un modulo nel kernel per nascondere le proprie attività, modificando opportune chiamate di sistema.

Questa funzione di Xen-VMI ha il compito di ricostruire la lista riferita dal simbolo *modules* per ottenere la lista dei moduli inseriti nel kernel. Successivamente, questa funziona verifica l'integrità di tutti i moduli e, inoltre, se essi sono moduli autorizzati. A questo scopo, prima che una MV possa essere utilizzata, carichiamo manualmente tutti i moduli del kernel di cui Linux ha un effettivo bisogno e, per ogni modulo, questa funzione calcola l'hash delle pagine in memoria contenenti il codice e salva questo valore, associato al nome del modulo, su file. Successivamente, quando la MV è in esecuzione, l'introspettore invoca questa funzione periodicamente per verificare che tutti i moduli in esecuzione siano moduli autorizzati e il cui codice non sia stato modificato, confrontando i valori degli hash calcolati in precedenza con quelli calcolati durante la nuova invocazione. Una differenza nei valori degli hash o nel nome implica o che un modulo autorizzato è stato modificato o che un modulo non autorizzato è stato inserito nel kernel.

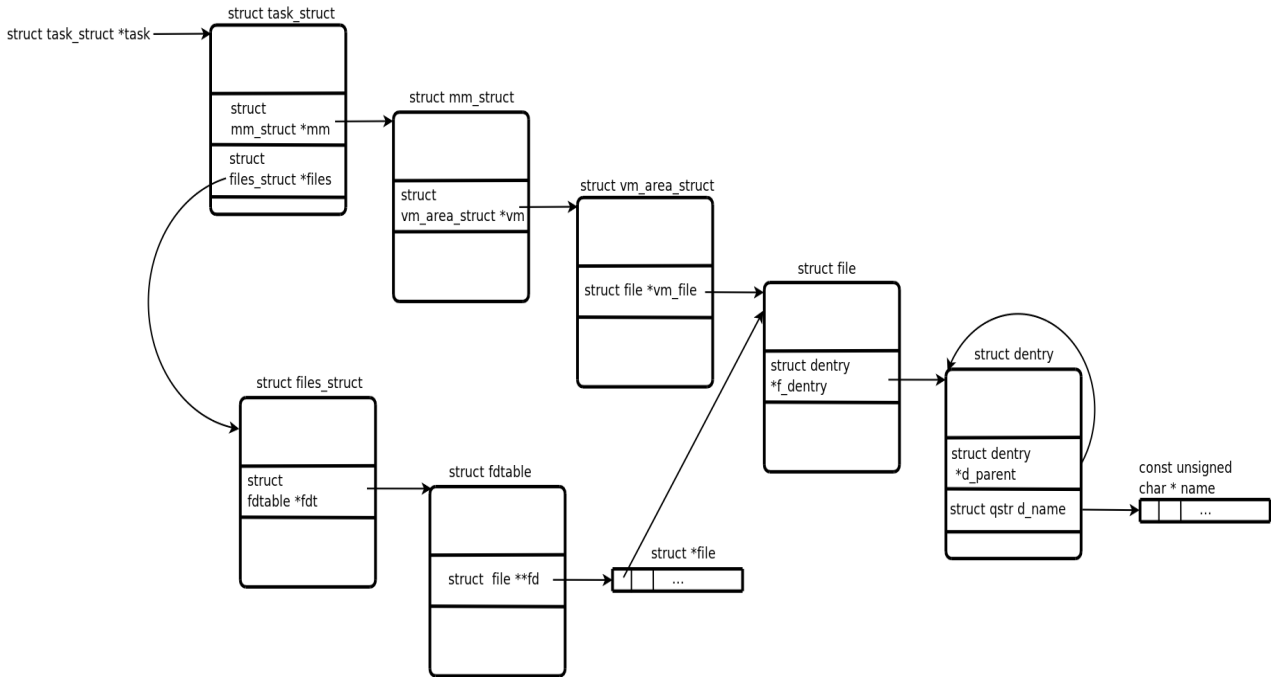

Figura 2: File Aperti di un Processo

Controllo dei Processi in Esecuzione

Questa funzione presente in Xen-VMI ricostruisce la lista puntata dal simbolo del kernel *init_task* e ottiene la lista dei processi in esecuzione su una Mon-MV. Successivamente, l'introspettore compara questa lista con quella ottenuta dal comando di utilità *ps* eseguito direttamente all'interno della Mon-MV. Se i due insiemi di identificatori di processo, con le relative informazioni (percorso dell'eseguibile, identificativo dell'utente proprietario del processo etc) sono diversi, allora è molto probabile che un attaccante abbia sostituito questo programma con una versione modificata per nascondere alcuni processi non autorizzati.

Controllo dei File Aperti

Analogamente alla funzione precedente, questa funzione ottiene la lista dei processi in esecuzione su una Mon-MV e, per ognuno di essi, ricostruisce: (i) la lista dei file che un processo ha aperto utilizzando la chiamata di sistema `open()`; (ii) la lista dei file mappati in memoria, ad esempio le librerie condivise (i file .so su Linux) che il caricatore di Linux carica nello spazio di indirizzamento di un processo all'avvio (o quando serve), oppure direttamente utilizzando la chiamata di sistema `mmap()`. La Fig. 2 mostra le strutture dati del kernel che devono essere ricostruite da Xen-VMI, a partire dai dati ottenuti tramite introspezione, per ottenere la lista dei file aperti.

Controllo della Modalità Promiscua.

Questa funzione di libreria ottiene le pagine del kernel che, a partire dal simbolo *dev_base*, contengono una lista di strutture dati relative alle interfacce di rete presenti sul sistema. Per ognuna di queste strutture, la funzione verifica se i flag relativi alla modalità promiscua sono settati. Questo approccio è lo stesso implementato da kstat [15], ma ad un livello distinto, inferiore. Infatti, kstat accede a queste strutture a livello utente tramite il device speciale */dev/kmem* o, analogamente, se implementato come modulo del kernel, a livello di kernel. Invece, questa funzione di libreria permette di applicare gli stessi controlli anti-promiscuità dal livello hardware/firmware. Per cui, diventa più complesso eludere questi controlli anche nel caso in cui l'attaccante dovesse ottenere i privilegi di root.

4 VALUTAZIONE DELLA SICUREZZA E DELLE PRESTAZIONI

In questa sezione descriviamo gli attacchi che sono stati portati contro il kernel e che sono rilevati utilizzando le tecniche descritte precedentemente. Inoltre, quantifichiamo l'entità dell'overhead dovuto all'applicazione periodica dei controlli di consistenza sul kernel delle Mon-MV.

4.1 Efficacia dei Controlli

Per testare l'efficacia dei controlli effettuati utilizzando la libreria Xen-VMI abbiamo effettuato i seguenti test:

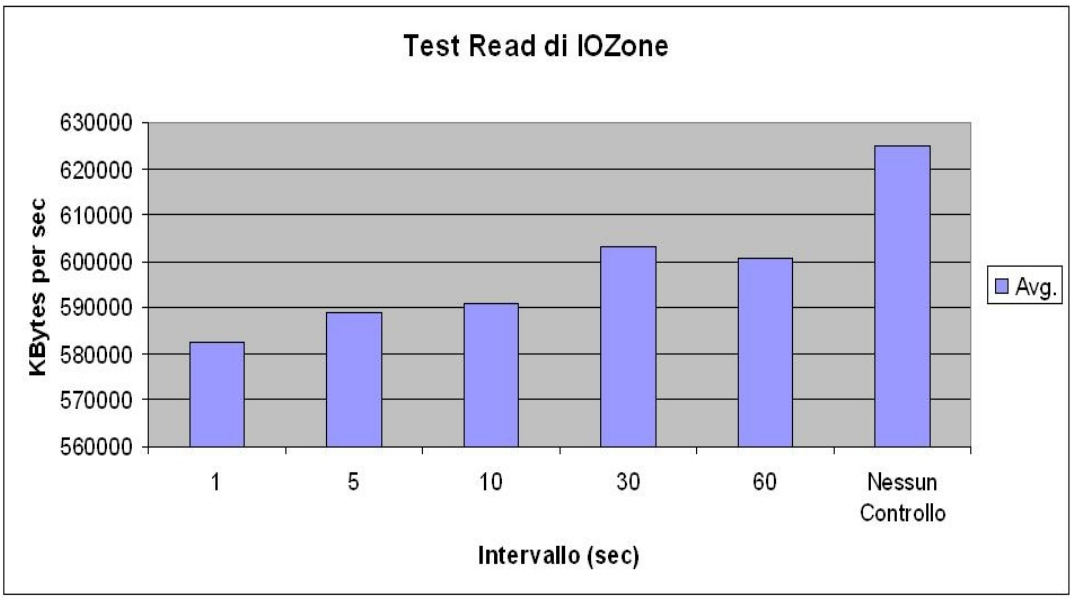

Figura 3: Performance

- abbiamo realizzato un semplice rootkit per modificare una entry nella tabella delle interruzioni *trap_table* [24];
- abbiamo inserito un modulo non autorizzato nel kernel di una Mon-MV per modificare sia un puntatore nella tabella *sys_call_table* che il codice di una chiamata di sistema;
- abbiamo sostituito i binari di sistema *ps* e *lsof* con versioni modificate per nascondere, rispettivamente, alcuni processi o file;
- abbiamo eseguito diversi sniffer che settano la modalità promiscua dell'interfaccia di rete delle Mon-MV. Ciò avviene tramite invocazione della chiamata di sistema `ioctl()` oppure `setsockopt()`. Da notare che nel primo caso l'utilità di sistema `ifconfig` rileva la modalità promiscua, nel secondo caso no.

L'introspettore, tramite le funzioni di libreria Xen-VMI, rileva sia le modifiche alle chiamate di sistema che all'handler delle chiamate di sistema (in entrambi i casi il codice si trova nella sezione text del kernel) che le modifiche ai puntatori nella tabella delle chiamate di sistema o nella tabella contenente le entry per le interruzioni. Inoltre, l'introspettore rileva sia che un modulo inserito non è autorizzato che la modifica al codice di un modulo autorizzato. Infine, l'introspettore rileva facilmente quando l'interfaccia è settata in modalità promiscua in entrambi i casi descritti.

4.2 Performance

Per quantificare l'overhead introdotto dai controlli di consistenza, abbiamo eseguito il benchmark IOzone Filesystem [21] su una Mon-MV mentre la I-MV esegue tutti i controlli di consistenza del kernel tramite l'utilizzo della libreria Xen-VMI. Il periodo che trascorre tra l'invocazione dei controlli è stato variato da 1 a 60 secondi e, inoltre, è stato fatto lo stesso benchmark disabilitando i controlli. La Fig. 3 mostra che, nel caso peggiore, la degradazione delle prestazioni sul test *read* dovuto ai controlli di consistenza è inferiore al 10% rispetto al caso in cui non sono effettuati i controlli. Lo stesso risultato è stato ottenuto per il test *write*. Ovviamente, anche Xen di per sé introduce un minimo overhead [7], da considerare se questi risultati sono confrontati con quelli ottenuti utilizzando la tecnica dell'introspezione a livello hardware/firmware tramite componenti specializzate.

4.3 Limiti Attuali del Prototipo

Al momento, ci sono alcuni attacchi che la I-MV non rileva tramite Xen-VMI. Ad esempio, ci sono altre regioni critiche nel kernel che devono essere protette, oltre alla tabella dei puntatori alle chiamate di sistema o alla tabella delle interruzioni, come le strutture dati del Virtual File System [36, 41] o altre [2, 5]. Inoltre, qualsiasi modifica illecita ai dati dinamici in memoria, come ad esempio allo stack del kernel, non è rilevata. La complessità di prevenire e rilevare questo tipo di attacchi è molto alta, perché è necessario calcolare a priori un insieme di invarianti rela-

tive a queste strutture dati, in modo che l'introspettore a tempo di esecuzione possa verificarne la loro validità. Un altro problema nasce se l'attaccante è a conoscenza del fatto che il sistema operativo è in esecuzione all'interno di una macchina virtuale [12]. Infatti, l'attaccante può tentare di attaccare direttamente il VMM oppure provare ad eludere i controlli periodici di consistenza. Nel caso dell'evasione dei controlli, un attaccante può modificare le strutture dati critiche del kernel non appena i controlli di consistenza sono stati eseguiti e successivamente ripristinare il sistema ad uno stato consistente prima che questi controlli vengano nuovamente effettuati, evitando così di essere rilevato. Infine, se un attaccante riuscisse a modificare il kernel associato ad una MV prima che essa sia messa in esecuzione, alcuni attacchi non sarebbero rilevati, ad esempio un modulo con finalità illecite potrebbe essere considerato autentico e quindi autorizzato ad essere eseguito.

5 STATO DELL'ARTE

La tecnica VMI è stata proposta dai ricercatori dell'università di Stanford in [18] assieme ad un prototipo di sistema per rilevamento delle intrusioni tramite VMI, chiamato *Livewire*. Il sistema *ReVirt* [10] supporta il *recovery*, *checkpoint* e il *roll-back* delle MV e utilizza la tecnica chiamata *virtual-machine replay* per ri-eseguire un sistema, istruzione per istruzione, all'interno di una MV, ad esempio per ripristinare il sistema dopo compromissioni. Inoltre, un'estensione a questo approccio è il sistema *Intro-Virt* [22] che rileva intrusioni su una MV effettuando controlli basandosi su un insieme di predicati relativi a vulnerabilità presenti nel codice. In questo modo, Introvirt può anche rilevare se le vulnerabilità sono state sfruttate in tempi passati. In maniera simile all'approccio proposto in questo paper, *XENKimono* [33] rileva violazioni della sicurezza su un kernel di un sistema operativo a tempo di esecuzione, utilizzando le tecnologie di virtualizzazione per effettuare i controlli attraverso una macchina distinta da quella monitorata. [41] presenta un framework per proteggere l'integrità del kernel, descrivendo sia le politiche che i meccanismi da implementare basandosi su un modello di logica evento-predicato-azione per specificare un insieme di politiche di sicurezza sull'utilizzo corretto del kernel. [23] propone una tecnica che è simile a quella proposta con Xen-VMI che è definita *guest view casting*, utilizzata per ricostruire le strutture dati del kernel tramite il VMM, per avere la stessa semantica del sistema operativo. Questa tecnica permette così di eseguire da un'altra macchina virtuale software anti-malware commerciale per rilevare violazioni alla macchina virtuale monitorata. *Paladin* [1] descrive un framework che, grazie alle tecnologie di virtualizzazione, riesce a rilevare e contenere gli attacchi effettuati tramite l'utilizzo di rootkit. *Manitou* [27] è un sistema realizzato nel VMM che assicura che una MV esegua solo codice autorizzato. Per fare questo, Manitou calcola l'hash di ogni pagina di memoria contenente il codice e setta il bit di esecuzione per la pagina solo se l'hash è contenuto in una lista di software autorizzato ad essere eseguito. Infine, [39] descrive un framework per la gestione gerarchica dell'affidabilità del software, dove la radice della gerarchia è un “secure co-processor” che, periodicamente, avvia una serie di gestori per effettuare controlli di sicurezza a più livelli così da realizzare una chain of trust.

6 CONCLUSIONI E SVILUPPI FUTURI

In questo paper abbiamo proposto una metodologia per rilevare attacchi contro il kernel di un sistema operativo in esecuzione su una MV, che sfrutta l'accesso diretto fornito dal VMM alle componenti allocate alla MV. Abbiamo discusso la progettazione e la realizzazione di Xen-VMI, una libreria di funzioni per monitorare l'integrità del kernel in esecuzione su una MV da un'altra MV tramite la tecnica della introspezione su macchine virtuali. Infine, abbiamo valutato l'overhead introdotto dai controlli di consistenza che, nel caso peggiore, è inferiore al 10%. Per ridurre ulteriormente l'overhead, uno sviluppo futuro prevede che, ogni volta che viene invocata una funzione Xen-VMI, si mantenga una tabella hash con la pagine già mappate, in modo da verificare efficientemente se un indirizzo virtuale del kernel è già presente in queste pagine, evitando di mappare più volte le stesse pagine. Ovviamente, questa tecnica può essere utilizzata in quanto la Mon-MV viene congelata prima di effettuare i controlli. Un'altra area su cui intendiamo lavorare si focalizza sullo studio di una metodologia per calcolare un insieme d'invarianti del kernel. Le invarianti devono essere calcolate applicando metodi formali di analisi statica del codice, ad esempio tramite interpretazione astratta [6, 4, 3]. Una volta calcolate le invarianti, queste sono date in input all'introspettore che, ad esempio, ogni volta che una chiamata di sistema viene invocata applica l'introspezione per controllare se le invarianti relative ai parametri della chiamate di sistema sono verificate.

BIBLIOGRAFIA

[1] Arati Baliga, Xiaoxin Chen, and Liviu Iftode. Paladin: Automated detection and containment of rootkit attacks, Jan 2006. Rutgers University Department of Computer Science Technical Report DCS-TR-593.

[2] Arati Baliga, Pandurang Kamat, and Liviu Iftode, Lurking in the shadows: Identifying systemic threats to kernel data, SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy (Washington, DC, USA), IEEE Computer Society, 2007, pp. 246–251.

[3] Thomas Ball, Rupak Majumdar, Todd D Millstein, and Sriram K Rajamani. Automatic predicate abstraction of c programs. In SIGPLAN Conference on Programming Language Design and Implementation, pages 203–213, 2001.

[4] Francois Bourdoncle. Abstract debugging of higher-order imperative languages. In PLDI '93: Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation, pages 46–55, New York, NY, USA, 1993. ACM Press.

[5] buffer, Hijacking Linux Page Fault Handler, Phrack 11 (2003), no. 61.

[6] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In POPL, pages 238–252, 1977.

[7] Bryan Clark, Todd Deshane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews, Xen and the art of repeated research, ATEC'04: Proceedings of the USENIX Annual Technical Conference 2004 on USENIX Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2004, pp. 47–47.

[8] chkrootkit – locally checks for signs of a rootkit. http://www.chkrootkit.org/.

[9] B. Dragovic, K Fraser, S Hand, T Harris, A Ho, I Pratt, A Warfield, P Barham, and R Neugebauer. Xen and the art of virtualization. In Proceedings of the ACM Symposium on Operating Systems Principles, October 2003.

[10] George W Dunlap, Samuel T King, Sukru Cinar, Murtaza A Basrai, and Peter M. Chen. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. In proc. of the 5th symposium on Operating systems design and implementation, pages 211–224, New York, NY, USA, 2002. ACM Press.

[11] P. Ferrie, Attacks on Virtual Machine Emulators, Symantec Security Response 5 (2006).

[12] Jason Franklin, Mark Luk, Jonathan M. Mc-Cune, Arvind Seshadri, Adrian Perrig, and Leendert Van Doorn.Towards sound detection of virtual machines.

[13] T Fraser. Automatic discovery of integrity constraints in binary kernel modules, Tech. report, University of Maryland Institute for Advanced Computer Studies, December 2004.

[14] The FU rootkit. http://www.rootkit.com/project.php?id=12.

[15] FuSyS. Kstat. http://www.s0ftpj.org/tools/kstat24 v1.1-2.tgz.

[16] Julian B. Grizzard, John G. Levine, and Henry L. Owen, Re-establishing trust in compromised systems: Recovering from rootkits that trojan the system call table, ESORICS, 2004, pp. 369–384.

[17] R P Goldberg. Survey of virtual machine research. IEEE Computer, 7(6):34–45, 1974.

[18] T Garfinkel and M Rosenblum. A virtual machine introspection based architecture for intrusion detection. In Proc. Network and Distributed Systems Security Symposium, February 2003.

[19] grsecurity, http://www.grsecurity.net/

[20] halflife, Abuse of the Linux Kernel for Fun and Profit, Phrack 7 (1997), no. 50.

[21] IOzone Filesystem Benchmark, http://www.iozone.org/.

[22] Ashlesha Joshi, Samuel T King, George W Dunlap, and Peter M Chen. Detecting past and present intrusions through vulnerability specific predicates. In SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles, pages 91–104, New York, NY, USA, 2005. ACM Press.

[23] X. Jiang, X. Wang, and D. Xu, Stealthy malware detection through vmm-based 'out-ofthe-box' semantic view reconstruction, Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS 2007), November 2007.

[24] kad. Handling Interrupt Descriptor Table for fun and profit. Phrack, 11(59), July 2002.

[25] John Levine, Julian Grizzard, and Henry Owen, A methodology to detect and characterize kernel level rootkit exploits involving redirection of the system call table, IWIA '04: Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04) (Washington, DC, USA), IEEE Computer Society, 2004, p. 107.

[26] LIDS project: LIDS Secure Linux System, http://www.lids.org/.

[27] Lionel Litty and David Lie. Manitou: a layer below approach to fighting malware. In ASID '06: Proceedings of the 1st workshop on Architectural and system support for improving software dependability, pages 6–11, New York, NY, USA, 2006. ACM Press.

[28] P A Loscocco and S D Smalley. Meeting critical security objectives with security enhanced linux. In Proc. of the 2001 Ottawa Linux Symposium, 2001.

[29] Peter Loscocco and Stephen Smalley. Integrating flexible support for security policies into the linux operating system. In Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pages 29–42, Berkeley, CA, USA, 2001. USENIX Association.

[30] Openssl: The open source toolkit for ssl/tls. http://www.openssl.org/.

[31] Open source tripwire. http://sourceforge.net/projects/tripwire/.

[32] Nick L Petroni, Timothy Fraser, Jesus Molina, and William A Arbaugh. Copilot - a coprocessor-based kernel runtime integrity monitor. In USENIX Security Symposium, pages 179–194, 2004.

[33] Nguyen Anh Quynh and Yoshiyasu Takefuji, Towards a tamper-resistant kernel rootkit detector, SAC '07: Proceedings of the 2007 ACM symposium on Applied computing (New York, NY, USA), ACM Press, 2007, pp. 276–283.

[34] S Sparks and J Butler. Shadow Walker: raising the bar for rootkit detection. www.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf.

[35] sd and devik. Linux on-the-fly kernel patching without LKM. Phrack, 10(58), December 2001.

[36] Alkesh Shah, Analysis of rootkits: Attack approaches and detection mechanisms, Tech. Report, Georgia Institute of Technology.

[37] University of Cambridge UK, Xen interface manual: Xen v3.0 for x86.

[38] VMware. http://www.vmware.com/.

[39] Lifu Wang and Partha Dasgupta. Kernel and application integrity assurance: Ensuring freedom from rootkits and malware in a computer system. In AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and ApplicationsWorkshops, pages 583–589,Washington, DC, USA, 2007. IEEE Computer Society.

[40] XenAccess Library. http://xenaccess.sourceforge.net/.

[41] Min Xu, Xuxian Jiang, Ravi Sandhu, and Xinwen Zhang, Towards a vmm-based usage control framework for os kernel integrity protection, SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies (New York, NY, USA), ACM Press, 2007, pp. 71–80.