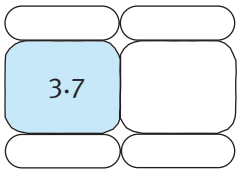
IL RUOLO DEL SOFTWARE SULL'INSICUREZZA DEI SISTEMI ICT

Danilo Bruschi

L'affermazione di Internet ha contribuito a far emergere il problema dell'insicurezza delle tecnologie ICT. Questo fenomeno è spesso riconducibile al software: errori presenti in esso o il suo "cattivo" uso determinano gran parte dei problemi di sicurezza. Per capire la natura del fenomeno "sicurezza informatica", e sapersi ad esso rapportare, è necessario capire come un errore presente in un programma possa essere utilizzato per compromettere la sicurezza del sistema su cui opera.



1. INTRODUZIONE

Nell'ultimo decennio, il mondo occidentale ha dovuto prendere atto, accanto all'affermazione delle tecnologie dell'informazione e della comunicazione (ICT), della "insicurezza" di tali sistemi. Il fenomeno già noto agli esperti a partire dai primi anni '60, ha assunto, con la diffusione delle rete Internet, dimensioni tali da non poter essere più circoscritto a un ristretto numero di persone ed è diventato argomento d'interesse per larghe fasce di utenti. Oggi, è difficile che ci sia un qualunque evento mediatico che affronta il tema delle tecnologie dell'informazione e della comunicazione senza considerarne gli impatti sulla sicurezza.

Ma precisamente che cosa si intende per tecnologia dell'informazione e della comunicazione sicura? Nel gergo comune, l'aggettivo *sicuro* denota principalmente due diverse proprietà di un sistema. Infatti, si dice che un sistema è sicuro quando, anche in presenza di malfunzionamenti, non provoca danni a chi lo usa (è, per esempio, in questa accezione che si parla di automobili sicure, o di apparecchi elettrici sicuri); oppure, un sistema è sicuro quando è in grado di preservare il proprio contenuto da accessi non autorizzati. È in questa accezione che, per esempio, si parla di una banca sicura, di una casa sicura ecc.. Nell'ambito informatico l'aggettivo *sicuro* viene usato con quest'ultima accezione. Quindi, un sistema informatico è sicuro quando protegge da "accessi non autorizzati" il proprio contenuto, cioè i dati in esso memorizzati e i servizi che eroga.

Da questa elementare considerazione discende la definizione universalmente accettata di sistema informatico sicuro. Tale definizione può essere così sintetizzata: *un sistema di calcolo viene considerato sicuro quando è in grado di garantire il soddisfacimento delle proprietà di confidenzialità, integrità e disponibilità*. Più precisamente, quando il sistema è in grado di garantire che: ogni utente può accedere esclusivamente alle informazioni di sua competenza (confidenzialità o riservatezza), ogni utente può modificare solo informazioni di sua competenza (integrità) e ogni azione intrapresa da

persone non autorizzate mirata a compromettere il corretto uso di una qualunque risorsa del sistema, è preventivamente bloccata (disponibilità). La sicurezza informatica è, invece, quella branca dell'informatica che studia e individua le tecniche e le metodologie per rendere le tecnologie ICT sempre più sicure.

Il problema della sicurezza delle tecnologie ICT è stato affrontato a partire dalla metà degli anni '60, sotto la spinta di due avvenimenti: l'avvento dei sistemi operativi multi-utente e l'introduzione massiccia dei sistemi di calcolo in ambiti militari. È ARPA (*Advanced Research Projects Agency*) l'agenzia per la ricerca del DoD (*Department of Defense*) statunitense ad avviare il primo progetto relativo alla costruzione di un sistema operativo sicuro: il **MULTICS**, che viene sviluppato a partire dai primi anni '60, al Massachusetts Institute of Technology (MIT) in collaborazione con Honeywell e General Electric. Alla base del progetto c'è la considerazione che se la condivisione delle risorse di un sistema di calcolo è prima di tutto una necessità economica, nel momento in cui più utenti di un calcolatore condividono memoria centrale, CPU (*Central Processing Unit*) e spazio su disco, è possibile per un utente interferire (accidentalmente o volontariamente) con le attività svolte da un altro utente. Diventa, quindi, prioritario individuare e realizzare una serie di meccanismi sia *hardware* che *software* (in particolare a livello di sistema operativo) per garantire che nell'ambito di sistemi condivisi da più persone, un utente sia vincolato a svolgere solo le operazioni di propria competenza.

Sullo slancio di queste motivazioni venne dato l'avvio a un'intensa attività di studio e ricerca svolta principalmente a livello accademico, mirata all'individuazione dei meccanismi più efficaci per la protezione dei dati e dei programmi nell'ambito dei sistemi e nell'individuazione di sistemi per la verifica formale dei programmi.

MULTICS è stato sottoposto a un processo di certificazione di sicurezza, secondo i criteri TCSEC (*Trusted Computing Security Evaluation Criteria*) proposti dal DoD. Nell'ambito di questo standard i sistemi informatici sono classificati secondo sette livelli di valutazione: D, C1, C2, B1, B2, B3, A1 in base alle funzionalità di sicurezza che sono in grado di offrire. D è il livello assegnato ai sistemi, che non offrono alcuna garanzia di protezione. Verso la fine degli anni '80 viene certificato che MULTICS soddisfa i requisiti imposti dal livello di certificazione B2. Una certificazione di sicurezza di tutto rispetto se si pensa che la configurazione di *default* di un attuale sistema operativo non soddisfa nemmeno i criteri di sicurezza previsti per il livello C2 e che il massimo livello di certificazione sinora ottenuto da un qualunque altro sistema operativo commerciale non ha mai superato B1.

I primi risultati di queste ricerche, in particolare, sono immediatamente applicati all'interno del sistema operativo MULTICS, che ancora oggi risulta essere il sistema operativo commerciale più sicuro che mai sia stato realizzato.

Già dall'esperienza MULTICS apparve evidente che la presenza in un sistema di calcolo di meccanismi di protezione non bastava da sola a garantire la sicurezza del sistema stesso. Infatti, i ricercatori che lavoravano a questo progetto constatarono immediatamente che il software che implementava questi meccanismi, o il software di sistema, su cui questi meccanismi tipicamente si appoggiano, conteneva errori commessi nelle fasi di progettazione e/o di programmazione, genericamente indicati come *security bug*. Questi errori consentivano, in determinate occasioni, di "bypassare" i sistemi di protezione rendendoli, quindi, del tutto inutili. Questo problema ha poi accompagnato tutti progetti software di un certo rilievo sviluppati sino ai nostri giorni, e ancora oggi è tra le cause principali dell'insicurezza dei sistemi informatici.

Non si è, quindi, di fronte a un problema determinato dalla mancanza di strumenti concettuali o tecnologici per la sua soluzione, ma *si è di fronte a un problema generato nella stragrande maggioranza dei casi da una incapacità di implementare correttamente soluzioni*[1]. Questo aspetto rende la realizzazione di *sistemi informatici sicuri* estremamente complessa, o meglio impossibile. Infatti, realizzare un sistema informatico sicuro significa, innanzitutto, realizzare un sistema il cui software non contenga security bug, e che sia correttamente installato. Purtroppo, ad oggi, non esiste alcun tipo di pro-

1 Sono esclusi da questo discorso i problemi di sicurezza legati alla non disponibilità dei sistemi, che solo rare volte sfruttano *security bug*.

cedimento manuale o automatico che consente la verifica *a priori* di tali proprietà, e non si intravede nemmeno la possibilità di realizzarlo in futuro. Quindi, l'unica cosa che resta da fare è prodigarsi per aumentare il livello di sicurezza dei sistemi attuali, con interventi mirati. Anche in questo caso vale comunque la pena ricordare che nonostante oggi si posseggano metodologie e strumenti per progettare sistemi informatici con un buon livello di protezione, la realizzazione degli stessi è sempre al di sotto delle aspettative. Il problema è sempre il solito: il *gap* che esiste tra il *dire* e il *fare* è nell'ambito della sicurezza fatale. Il fattore umano (in termini di conoscenza, esperienza ma anche limiti naturali) gioca un ruolo determinante nella realizzazione di questi sistemi.
Capire quali sono i presupposti affinché tutto ciò possa verificarsi e come un errore presente in un programma possa stravolgere i principi di funzionamento di un sistema significa capire quali sono i fondamenti su cui si basa l'(in)sicurezza dei sistemi informatici, e, quindi, possedere le nozioni necessarie per poter correttamente affrontare il problema. In questo articolo si cercherà di fornire al lettore le nozioni necessarie per poter cogliere questi elementi. Non si parlerà, quindi, delle metodologie e tecnologie per realizzare sistemi sicuri, ma delle principali cause del fenomeno, convinti che solo una conoscenza approfondita delle stesse sia un fattore abilitante per l'individuazione di soluzioni efficaci anche se solo parziali.

2. LE DIMENSIONI DEL PROBLEMA

Quali sono le conseguenze di un security bug? Quanti sono i security bug noti fino ad oggi? Sono queste alcune delle domande necessarie per poter quantificare correttamente il problema sopra descritto. Ad oggi, sono circa 10.000 [3] le vulnerabilità note per poter aggirare i meccanismi di protezione dei diversi sistemi informatici che popolano il nostro pianeta. Ognuna di queste vulnerabilità può dare origine ad una o più tecniche di attacco o di intrusione (per una definizione precisa di questi **termini** si veda il riquadro), che consentono lo svolgimento sul sistema attaccato di diverse operazioni non autorizzate. In particolare, in riferimento alle attività che un utente è in grado di compiere, le tecniche di attacco informatico sono raggruppabili in 8 diverse categorie riportate nella tabella 1.

3. L'AFFIDABILITÀ DEL SOFTWARE

Come è stato anticipato la principale causa dell'insicurezza dei sistemi informatici è riconducibile ai security bug. Questi errori possono essere stati compiuti in ciascuna delle fasi che costituiscono il ciclo di vita del software: disegno o progettazione, programmazione, test e installazione. Un errore in una di queste fasi si riflette inesorabilmente nella "messa in opera" di un prodotto "vulnerabile", che in particolari circostanze consente a un intrusore di compiere sul sistema attività non autorizzate. Si vedano, nel seguito, quali sono i tipici errori che possono essere commessi in ciascuna di queste fasi.
In fase di disegno o progettazione di un programma è necessario definire il *thread model*, cioè una descrizione semi-formale di tutti i possibili comportamenti scorretti che il programma potrebbe originare. Conseguentemente, il programma deve essere progettato per poter far fronte a ciascuno di

Quando si parla di sicurezza informatica, **termini** come incidente, attacco, vulnerabilità e minaccia sono usati molto frequentemente e spesso, in contesti diversi, lo stesso termine viene ad assumere significati diversi. È quindi estremamente importante quando si affrontano questi problemi definire con precisione il significato di questi termini. In questo contributo, si è deciso di rifarci a uno sforzo di standardizzazione sul significato degli stessi, in corso nella comunità scientifica e i cui contributi di riferimento sono:
attacco o intrusione: una serie di attività che possono essere svolte su un sistema per poter svolgere da un intrusore per poter svolgere attività non autorizzate;
attaccante: un individuo che esegua uno o più attacchi per poter raggiungere i propri obiettivi;
evento: un'azione diretta verso un determinato obiettivo, con l'intento di modificarne lo stato;
IT security incident: ogni azione avversa rivolta contro un sistema IT, con l'intento di raggiungere uno dei seguenti obiettivi: violazione della confidenzialità/integrità dei dati, compromissione del livello di disponibilità dei servizi;
vulnerabilità: una falla o debolezza presente in un sistema che consente di eseguire sullo stesso un'azione non autorizzata.

Tipologia dell'incidente	Definizione	Esempi di tecniche utilizzate[1]	Potenziali effetti ottenibili
Computer Fingerprinting	Attività svolte al fine di raccogliere informazioni in merito a un host	Probing e scanning	Elenco dei servizi disponibili sull'host vittima e sue caratteristiche
Codice Maligno	Compromissione di un host attraverso l'esecuzione di programmi indipendenti	Virus, Trojan, spyware	Violazione della riservatezza e integrità dei dati e indisponibilità dei servizi. Abuso dei sistemi
Denial of service	Accessi continui a un servizio ai fini di saturarne le risorse	SYN-flood, Ping of Death, Land, WinNuke, TFN, TFN2K, Trin00, Slice3	Messa fuori uso, temporanea, del servizio attaccato
Account Compromise	Accesso non autorizzato a un sistema o alla risorsa di un sistema, in qualità di amministratore di sistema o di utente	Buffer overflow, Format bug, o uso di credenziali di accesso (username e password)	Violazione della riservatezza e integrità dei dati e indisponibilità dei servizi. Abuso dei sistemi
Accesso non autorizzato alle informazioni	Accessi non autorizzati lettura e/o scrittura dati	SQL-injection, Spyware	Violazione della riservatezza dei dati
Accesso non autorizzato al canale di comunicazione	Interferenze senza le necessarie autorizzazioni alla trasmissione di dati	Hijacking, replay attack, sniffing, ARP poisoning	Violazione della riservatezza e dell'integrità dei dati trasmessi in rete
Accesso non autorizzato a sistemi di comunicazione	Uso non autorizzato di sistemi e protocolli per la comunicazione	DNS spoofing, mail relays, war driving	Violazione della riservatezza di dati e accesso non autorizzato ai sistemi
Modifica non autorizzata di informazioni	Modifica di dati presenti su un computer senza le necessarie autorizzazioni	Web defacements, viruses, SQL-injection	Violazione dell'integrità dei dati

[1] Il lettore interessato ad approfondire le diverse tecniche può consultare il sito web riportato in bibliografia [3].

TABELLA 1
Tassonomia delle tecniche di intrusione informatica, in base alle attività svolte sul sistema vittima

questi attacchi. L'omissione di questa fase o una noncuranza durante il suo svolgimento porta a definire un *thread model* incompleto e, quindi, a progettare un prodotto che contiene delle vulnerabilità. Nel prossimo paragrafo verrà illustrato un esempio di errori di questo tipo, che ha consentito la realizzazione di un attacco informatico, ancora oggi molto praticato.
In fase di programmazione possono essere fatte da parte del programmatore delle scelte errate nell'implementazione di alcuni algoritmi, oppure essere utilizzate istruzioni che consentono lo svolgimento di attacchi come il *buffer overflow* che sarà descritto nel paragrafo 5. Per esempio, un baco presente nel programma di sistema *finger*, nell'ambito del sistema operativo SunOS, ha consentito nel 1988 la realizzazione del più famoso attacco informatico avve-

nuto su Internet e noto come **Internet Worm**. Nel 2001, un errore di programmazione commesso nella *routine* che interpretava le stringhe di *input* nel programma *Internet Information Server* (IIS, il *web server* di Microsoft), consentiva a un qualunque utente di prendere il controllo del sistema su cui era in esecuzione il programma.

Il 3 novembre del 1988 Robert Morris, uno studente di Ph.D della Cornell University, attraverso un programma da lui scritto, riuscì a guadagnare l'accesso e a mettere fuori uso circa 6000 calcolatori operanti in Internet. L'attacco noto come **Internet Worm** mise fuori uso calcolatori appartenenti a università, laboratori di ricerca, enti governativi e industrie. La notizia ebbe una vasta eco da parte dei *media* di tutto il mondo, che così prendeva consapevolezza del problema della sicurezza informatica.

In generale, i bachi di sicurezza sono difficilmente individuabili, poiché difficilmente influenzano il comportamento di un programma. Un errore di sicurezza difficilmente è causa di calcoli errati, visualizzazioni errate, o di messaggi di errori. Un programma che contiene errori di sicurezza può essere perfettamente aderente alle specifiche funzionali per cui è stato concepito. Diventa, quindi, molto difficile individuare security bug e solo un'accurata fase di test può rivelarne la presenza all'interno del codice. In fase di test tutte le possibili alternative offerte dal programma dovrebbero essere verificate, in particolare a partire da un thread model è necessario verificare il comportamento del programma quando sottoposto a tutti i possibili tentativi di attacco. La dimensione dei programmi e la necessità di distribuire a ritmi sempre più sostenuti nuove versioni di software per poter reggere il passo della concorrenza e i costi da sostenere, fanno sì che la fase di test sia però generalmente svolta in modo molto approssimativo con inevitabili ricadute sulla sicurezza del prodotto finale. Nell'ambito di progetti che godono di una certa criticità è invalsa l'abitudine di coinvolgere nelle fasi di test quelli che in gergo sono chiamati *Tiger Team* o *Red Team*. Si tratta di gruppi di esperti che tentano di forzare i sistemi di protezione di un prodotto software con l'obiettivo di individuarne eventuali falle. I tiger team sono solitamente composti da personale esterno e complementano così l'attività di test.

Un ulteriore elemento di criticità per la sicurezza dei sistemi è la fase di installazione del software. In questa fase, infatti, possono essere compiuti, da utenti poco esperti, una serie di errori che possono essere sfruttati per accedere abusivamente ai sistemi. Per esempio, ancora oggi, uno degli errori più comuni in fase di installazione di un sistema operativo è la creazione di utenze senza *password*, che consentono, quindi, di accedere al sistema senza particolari controlli facilitando gli accessi non autorizzati. Questo tipo di problema ha, come quelli precedentemente accennati, una forte incidenza sulla sicurezza dei sistemi informatici. Per ragioni di spazio ci si concentrerà, nei prossimi due paragrafi, sui problemi che si ritengono concettualmente più importanti: ovvero, gli errori commessi in fase di progettazione e programmazione.

4. ERRORI DI PROGETTAZIONE

In questo paragrafo, verrà descritto un attacco portato al protocollo ARP (*Address Resolution Protocol*) e noto come ARP Poisoning. L'attacco è ancora oggi molto diffuso e deriva da una serie di scelte errate commesse in fase di progettazione. Lo scopo di questo paragrafo è fornire un esempio concreto di come errori commessi in fase di progettazione del software, si ripercuotano nelle fasi successive di vita del software sino a comprometterne il corretto funzionamento.

In breve, il protocollo ARP è utilizzato in tutte le reti locali che adottano il **protocollo Ethernet** (circa il 98% di tutte le reti locali). Tale protocollo prevede che tutti gli *host* della LAN siano identificati attraverso un indirizzo numerico di 48 bit noto anche come MAC *address*. Le informazioni che i vari host di una LAN devono trasmettersi sono racchiuse in uno o più pacchetti, contenti il MAC address del destinatario e inviati sulla rete. In ricezione quando un host sulla LAN riceve un pacchetto verifica se l'indirizzo presente nel pacchetto è il

Generalmente, un host possiede tanti identificativi quante sono le reti diverse a cui è connesso. Nel caso della LAN basate sul **protocollo Ethernet** (adottato nell'ambito dello standard IEEE 802.3), l'identificativo è un numero di 48 bit, espresso come una sestupla di coppie di cifre esadecimali (per esempio, FF-00-23-78-AB-11). Questo identificativo non deve essere confuso con l'indirizzo IP, che serve, invece, per identificare un host sulla rete Internet. Un indirizzo IP è un numero di 32 bit, che viene espresso come una quadrupla di numeri compresi tra 0 e 255. All'indirizzo IP può anche essere associato un indirizzo simbolico del tipo nome@dico.unimi.it

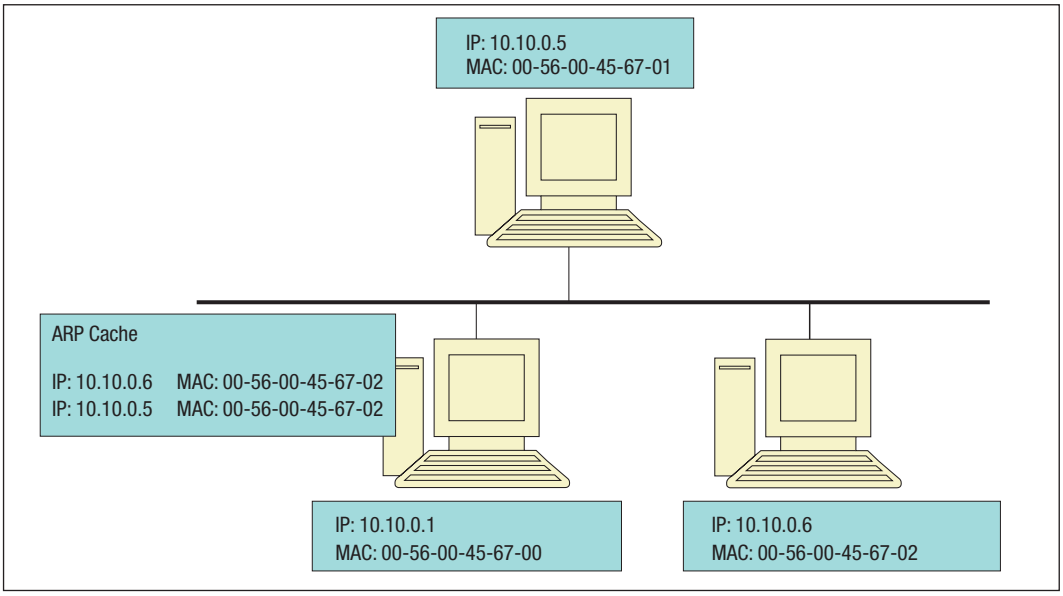


FIGURA 1
Schema di una semplice rete locale e di un attacco di ARP poisoning

suo, e in tal caso elabora le informazioni contenute, altrimenti lo elimina.
In genere, però il MAC address di un host non è noto, ma è noto solo il suo indirizzo IP (*Internet Protocol*) o indirizzo simbolico. È il protocollo ARP che nell'ambito di una LAN si preoccupa, dato l'indirizzo simbolico di un host, di individuarne il corrispondente MAC address[2]. Per esempio, si supponga che un host il cui indirizzo simbolico è host1@rete.dominio.it voglia comunicare con host2@rete.dominio.it, presente sulla stessa LAN. Prima che questa comunicazione avvenga, il protocollo ARP in esecuzione su host1@rete.dominio.it invia un messaggio a tutti gli host presenti sulla LAN (ARP Request), chiedendo all'host, il cui indirizzo simbolico è host2@rete.dominio.it, di fargli conoscere il proprio MAC address. Tra tutti gli host presenti sulla rete solo host2@rete.dominio.it risponderà a questo messaggio, e host1@rete.dominio.it provvederà a memorizzare le risposte ricevute, in una opportuna tabella chiamata *ARP cache*. Per ovvie ragioni di efficienza ogni host prima di inviare un *ARP request* consulta la propria ARP *cache* per verificare se il MAC address ricercato sia già presente.
ARP è stato progettato in modo da procedere all'aggiornamento dei valori presenti nella propria cache anche se non esplicitamente richiesti. Vale a dire, un host può di sua spontanea volontà, informare un altro host che il suo indirizzo MAC è stato modificato. L'host che riceve quest'informazione provvede a eseguire l'aggiornamento dell'informazione ricevuta, senza alcuna verifica sulla sua validità. Questa scelta progettuale dettata dalla necessità di realizzare un protocollo estremamente efficiente che prontamente si adatta alle modifiche della rete, è però anche una vulnerabilità che opportunamente sfruttata consente l'effettuazione del seguente attacco noto come ARP Poisoning.
Lo schema dell'attacco è abbastanza semplice. Si consideri la rete locale in figura 1. Si supponga che l'utente che opera sull'host 10.10.0.6 voglia intercettare tutto il traffico di rete diretto dall'host 10.10.0.1 all'host 10.10.0.5. Sfruttando le "proprietà" del protocollo ARP deve limitarsi a inviare un messaggio ARP per l'aggiornamento della cache all'host 10.10.0.1, e comunicargli che il MAC address di 10.10.0.5 è diventato 00-56-00-45-67-02. Quando l'host 10.10.0.1 riceverà questo messaggio aggiornerà il contenuto della propria cache, e ogni volta che dovrà inviare un messaggio all'host 10.10.0.5 lo invierà all'host che sulla rete locale ha l'indirizzo 00-56-00-45-67-02 quindi all'host 10.10.0.6, che raggiunge così l'obiettivo inizialmente preposto.

2 Qualora l'host ricercato non sia presente sulla LAN, il MAC address che verrà fornito dall'ARP è quello dell'Internet Gateway che provvederà a inoltrare il pacchetto sulla rete Internet.

Recenti studi sono stati intrapresi per cercare di ovviare a questa vulnerabilità e diverse soluzioni sono state proposte dalla comunità scientifica [2]. Tutte queste soluzioni presuppongono una o più *patch* a livello del *kernel* di sistema operativo: ciò significa che affinché queste soluzioni possano essere adottate universalmente devono essere incluse nelle distribuzioni ufficiali dei sistemi operativi. Nessun costruttore sembra però interessato al problema e le LAN continuano a essere soggette a questo tipo di attacco.
Quello che è stato appena descritto è un esempio emblematico dei problemi legati alla sicurezza dei calcolatori. Di protocolli vulnerabili come ARP ve ne sono molti, via via che sono individuati danno origine a nuove tecniche di attacco, che restano efficaci sino a che le corrispondenti patch correttive non sono individuate. Va comunque segnalato che per correggere errori di questa natura è necessario rivisitare il protocollo originale, e le modifiche in genere richieste sono abbastanza radicali (e possono a loro volta contenere errori). Non va poi dimenticato che il tempo che intercorre tra il rilascio delle correzioni e la loro messa in opera da parte degli utenti finali può anche essere dell'ordine di decine di mesi, in questo periodo, quindi, i sistemi restano comunque esposti a vulnerabilità note.

5. ERRORI DI PROGRAMMAZIONE

Tra le tecniche di attacco informatico più diffuse va sicuramente annoverato il *buffer overflow* introdotto da Morris con l'Internet Worm. Questo attacco usa delle peculiarità del linguaggio di programmazione C in cui è scritto gran parte del codice di sistema sia in ambito Unix che in ambito Windows. In particolare, il buffer overflow sfrutta il fatto che il compilatore C non controlla che in un'operazione di trasferimento dati la variabile sorgente abbia una dimensione superiore a quella di destinazione. In fase di esecuzione questa anomalia si traduce nel fatto che i dati superflui della variabile sorgente verranno scritti nelle zone di memoria circostanti la variabile di destinazione.
Per esempio, se all'interno di un programma

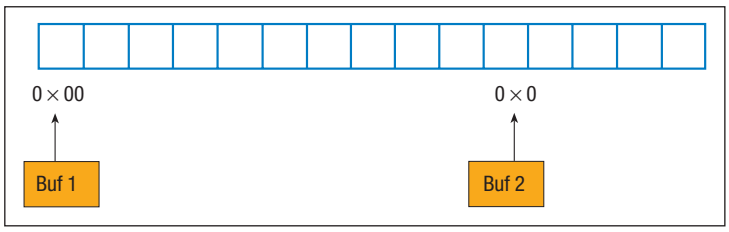


FIGURA 2
Schema di allocazione della memoria per le variabili Buf1 e Buf2

C sono presenti le seguenti istruzioni dichiarative:

```
char Buf1 [10];

char Buf2 [7];
```

il compilatore provvederà ad assegnare due aree di memoria contigue, in particolare poiché la dichiarazione di Buf1 precede quella di Buf2, a Buf1 saranno assegnate locazioni di memoria con indirizzi immediatamente inferiori a quelli assegnati alle locazioni destinate a contenere Buf2. Schematicamente la mappa di memoria può essere rappresentata nel modo indicato in figura 2.
Si supponga ora che, sempre all'interno dello stesso programma, si trovi la seguente istruzione di assegnamento:

```
Buf1 = "zzzzzzzzzzzz";
```

è facile constatare che con questa istruzione si tenta di inserire una stringa di 12 caratteri in una zona di memoria (Buf1) predisposta a contenerne 10.
Se, invece, del linguaggio C si stessero considerando altri linguaggi di programmazione come C++, Java, Pascal, C#, il compilatore rileverebbe questa inconsistenza e darebbe un messaggio di errore. Nel caso del compilatore C, invece, il problema non viene rilevato, anzi il compilatore provvede a recuperare lo spazio mancante dalle variabili contigue a Buf1. Più precisamente, la suddetta istruzione di assegnamento viene, schematicamente parlando, tradotta nel seguente modo:

```
i = indirizzo iniziale di Buf1;

j = 1;
```

while (l'elemento della stringa di dati da cari-

care in Buf1 di posizione *j* è diverso dal carattere di fine stringa)

Buf1 [i] = *j*-esimo elemento della stringa di input

i = *i* + 1;

j = *j* + 1;

endwhile

Questo significa che quando la suddetta istruzione di assegnamento sarà eseguita, la situazione che verrà a crearsi in memoria sarà quella rappresentata in figura 3.

Per esempio, la stringa http://www.bibliography.it/airchronicles/aureview/1979/jan-feb/schell.html digitata da un utente per accedere a un dato sito Web, viene memorizzata in un'apposita area di memoria del *browser*. Ovviamente, se nel browser non era stato predisposto un buffer di dimensioni opportune per ospitare le stringhe di input, la suddetta stringa ricoprirà i dati delle zone di memoria ad essa contigue.

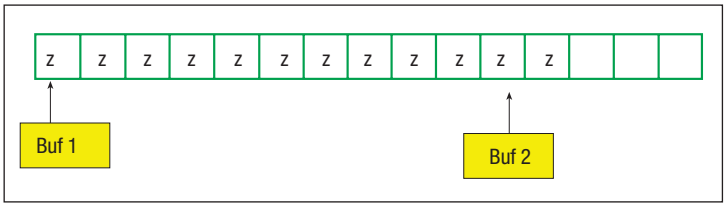


FIGURA 3

Configurazione dei due buffer dopo l'esecuzione dell'istruzione di assegnamento

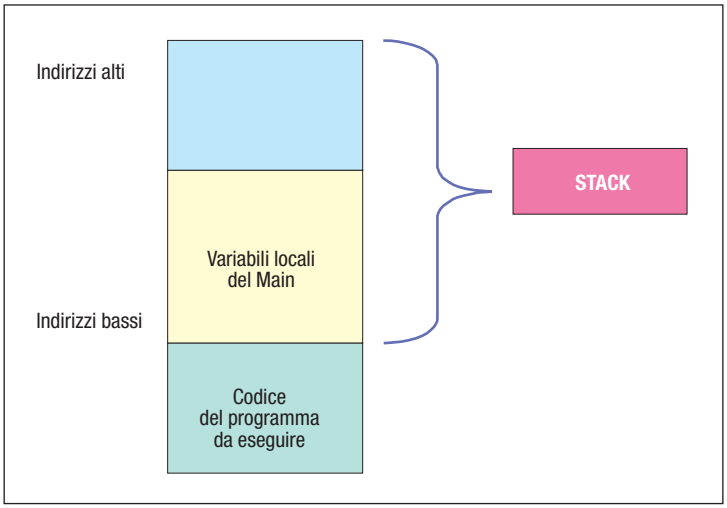


FIGURA 4

Schema di allocazione di un programma oggetto in memoria

Si veda nel seguito come questa caratteristica del C può essere utilizzata per compiere un attacco a un programma. Per fare ciò è necessario richiamare alcune nozioni relative alle modalità con cui viene caricato in memoria un programma per poter essere eseguito. Innanzitutto, si ricorda che un programma eseguibile è composto da due parti principali: una parte codice, che contiene le istruzioni da eseguire e una parte dati, che contiene i dati su cui il codice deve operare. Quando un programma deve essere eseguito sono caricati in memoria centrale le componenti codice e dati relative alla procedura principale (*main*). La parte dati, in particolare, viene caricata in uno *stack* i cui indirizzi decrescono, vengono cioè allocate prima le zone dello stack con indirizzi alti. Lo schema generale è riportato in figura 4.

Quando durante l'esecuzione del programma principale viene richiamata una procedura "secondaria", sullo stack viene salvato l'indirizzo di rientro al programma principale, cioè l'istruzione del programma principale che dovrà essere eseguita al termine dell'esecuzione della procedura secondaria, seguito dai dati relativi alla procedura stessa. Quindi, la situazione dello stack diventa quella riportata in figura 5.

Si supponga ora che la procedura secondaria usi una variabile di 512 caratteri per memorizzare dei dati di input. Tale variabile sarà allocata sullo stack e se nella fase di input venisse inserita una stringa più lunga di 512 caratteri, la stessa andrà a sovra scrivere (verso l'alto) le variabili adiacenti e paradossalmente questa operazione potrebbe essere effettuata dall'utente sull'indirizzo di ritorno. In questo caso, terminata l'esecuzione della procedura secondaria si passerebbe a eseguire l'istruzione il cui indirizzo è contenuto nella zona "riservata" all'indirizzo di ritorno che è però stato precedentemente sovra scritto, e quindi il programma originale non potrebbe continuare correttamente l'esecuzione. Se però, invece, di una stringa casuale l'utente avesse inserito una stringa contente il codice eseguibile di un programma, e avesse inserito l'indirizzo d'inizio di questo programma al posto dell'indirizzo di ritorno l'effetto otte-

nuto sarebbe stato decisamente diverso (Figura 6).
Nel momento in cui la procedura secondaria terminasse la propria esecuzione, verrebbe eseguita l'istruzione il cui indirizzo si trova nella zona di memoria riservata all'indirizzo di rientro e, quindi, il controllo invece che al programma originale sarebbe ceduto al programma scritto dall'utente. Questo programma potrebbe, per esempio, cancellare alcuni *file* dell'utente dal disco, modificare informazioni e nei casi peggiori arrivare anche a cancellare l'intero contenuto del disco fisso.
L'attacco è, ovviamente, molto difficile da realizzare, e richiede conoscenze molto approfondite delle architetture e dei sistemi operativi corrispondenti. In particolare, si è volutamente tralasciato una serie di particolari perché scopo di questo articolo è quello di consentire ai lettori di cogliere gli aspetti più importanti della tecnica.
Con questa tecnica sono stati compromessi i servizi di rete e i comandi più importanti di tutti i sistemi operativi. Diverse tecniche sono state finora proposte per far fronte a questo problema da parte della comunità internazionale [1]. Ancora oggi il buffer overflow è la tecnica più utilizzata per attaccare i sistemi e nonostante di questa tecnica si conosca ogni dettaglio realizzativo sono ancora molti i programmi di sistema che con questa tecnica sono e possono essere attaccati. Per esempio, il **worm Slammer** che nella primavera di quest'anno ha infettato più di 75000 host era basato su un attacco

Slammer (citato da alcuni autori anche come *Sapphire*) è stato il computer **worm** più veloce sinora realizzato. Slammer è "apparso" sulla rete il 25 gennaio 2003, e sfruttava attraverso il buffer overflow una vulnerabilità presente nei programmi Microsoft SQL Server e Microsoft SQL Server Desktop Engine (MSDE) 2000. La vulnerabilità era stata individuata nel Luglio 2002 e le corrispondenti patch correttive erano state rilasciate immediatamente dopo. Si stima che complessivamente Slammer abbia infettato più di 75.000 sistemi in tutto il mondo. La sua peculiarità è stata comunque la velocità di infezione. L'infezione della stragrande maggioranza dei sistemi è avvenuta nei primi dieci minuti di attività del virus, che ha poi rallentato la sua velocità di propagazione per errori presenti nel codice del virus stesso. A titolo di paragone si rammenta che prima di Slammer, la palma di virus più veloce era posseduta da Code Red, che era riuscito a infettare 359.000 sistemi in poco più di 19 ore.

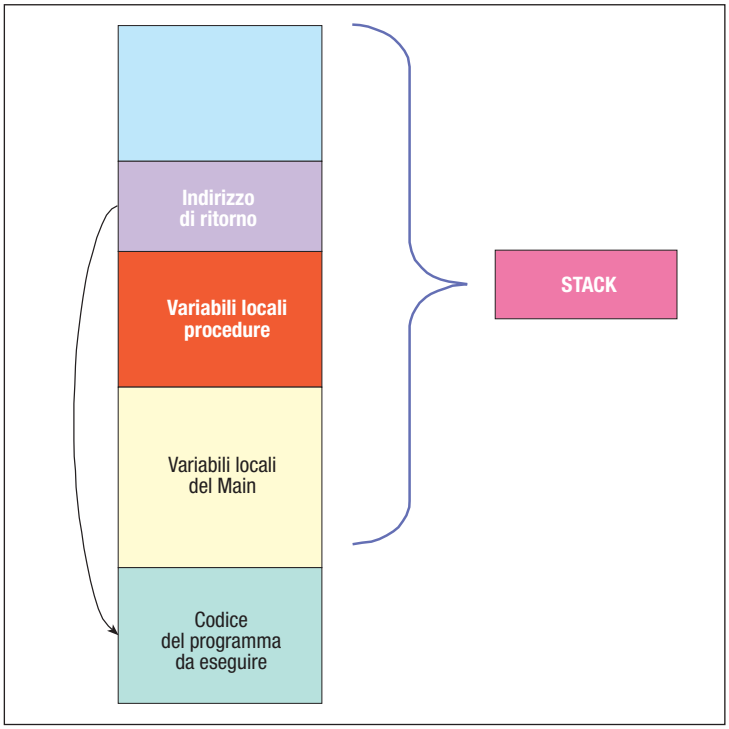


FIGURA 5
Schema di allocazione di un programma nella memoria centrale, dopo la chiamata di una procedura secondaria

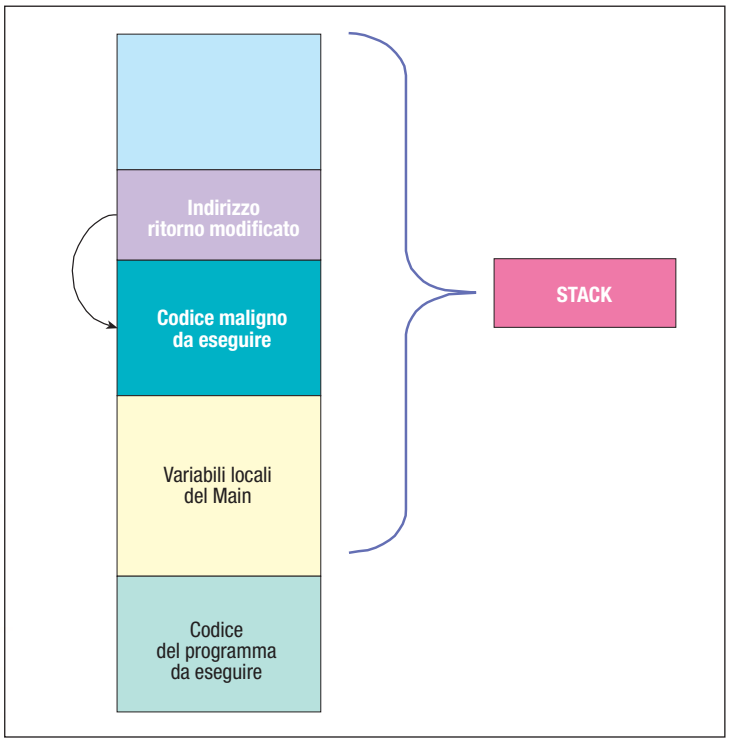


FIGURA 6
Schema di modifica della configurazione dello stack per consentire l'esecuzione di codice estraneo a quello del programma originale

di questo tipo applicato a una componente dei sistemi operativi Windows 2000 e Windows XP.

5. CONCLUSIONI

Con questo contributo l'autore spera di avere consentito al lettore di cogliere come la natura del problema sicurezza informatica sia da ricercarsi nei limiti naturali di chi su questi sistemi opera, e che nelle fasi di progettazione, realizzazione e uso dei programmi e delle tecnologie ICT commette inesorabilmente errori, solitamente dettati da una parziale conoscenza del problema o sottovalutazione di alcuni aspetti. Questo significa che la sicurezza dei sistemi ICT non può essere raggiunta attraverso l'adozione di opportune tecnologie o metodologie. Queste possono semmai contribuire ad alleviare il problema ma non a risolverlo. In realtà, chi scrive è fermamente convinto che esisterà sempre un problema sicurezza informatica, anche se l'adozione di nuove tecnologie potrebbe in un futuro non molto lontano consentire la realizzazione di sistemi molto più robusti di quelli attuali.

Bibliografia

[1] Bruschi D., Rosti E.: *A tool for pro-active defense against the buffer overrun attack*. Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, 2003.

[2] Bruschi D., Ornaghi A., Rosti E.: *Secure-ARP*. Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, 2003.

[3] Cert: http://www.cert.org/stats/cert_stat.html#vulnerabilities

[4] McLean J., "Security models". In Marciniak J: *Encyclopedia of Software engineering*. John Wiley &Sons, New York 1994.

[5] RFC2350: *Best Current Practice*.

[6] RFC2828: *Internet Security Glossary* by R. Shirey. May 2000.

[7] Schell R.G.: *Computer Security: the Achilles' heel of the electronic air force*. http://www.airpower.af.mil/airchronicles/aureview/1979/jan-feb/schell.html

[8] John D. Howard, Thomas A. Longstaff: *A Common Language for Computer Security Incidents*. Sandia National Laboratories.

DANILO BRUSCHI è professore ordinario di Sicurezza delle reti e dei calcolatori presso il Dipartimento di Informatica e Comunicazione dell'Università degli Studi di Milano. È membro del Comitato Tecnico Nazionale per la Sicurezza Informatica e delle Telecomunicazioni per la Pubblica Amministrazione. È socio fondatore e Presidente dell'Associazione Italiana per la Sicurezza Informatica (CLUSIT).
bruschi@dsi.unimi.it