# VIII JORNADAS STIC CCN-CERT

La defensa del patrimonio tecnológico frente a los ciberataques

10 y 11 de diciembre de 2014

CCN-CERT centro criptológico nacional

# Ghost in the Shell



www.ccn-cert.cni.es


**Jaime Sánchez (@segofensiva)**

**http://www.seguridadofensiva.com**

**jsanchez@seguridadofensiva.com**

TELEFÓNICA

# Índice

# INTRODUCCIÓN

- La base de datos de vulnerabilidades nacionales de Estados Unidos ha concedido a esta vulnerabilidad el valor máximo de severidad, con un **CVSS de 10/10.** Esta vulnerabilidad ha sido rápidamente comparada por los expertos en seguridad con otra Heartbleed, otra de alto riesgo que afectó hace unos meses al protocolo OpenSSL, por su severidad, debido a que permitía la extracción de bloques de información de la memoria en aquellos sistemas vulnerables.

- Shellshock es mucho más grave que HeartBleed ya que explotar estar vulnerabilidad nos permite la ejecución de código arbitrario, lo cual es mucho más grave que extraer información arbitraria de la memoria.

- A esto hay que sumarle que la complejidad de explotación de esta vulnerabilidad es muy baja, consistiendo básicamente en un payload en el que indicaremos en el valor de aquellas variables susceptibles a ser interpretadas por Bash incorrectamente

# VARIANTES DE SHELLSHOCK

- La vulnerabilidad inicial reportada al equipo de Bash fue denominada con el CVE-2014-6271. El error se corrigió con un parche para el programa, sin embargo, después del lanzamiento del parche, hubo informes posteriores de diferentes vulnerabilidades relacionadas:

  *env x='() { :;}; echo vulnerable' bash -c "echo this is a test"*

- CVE-2014-6277:

  *bash -c "f() { x() { _;}; x() { _;} <<a; }" 2>/dev/null || echo vulnerable*

- CVE-2014-6278:

  *shellshocker='() { echo You are vulnerable; }' bash -c shellshocker*

- CVE-2014-7169: descubierta por el consultor *Tavis Ormandy*, mientras se encontraba trabajando en la vulnerabilidad CVE-2014-6271:

  *env X='() { (a)=>\' sh -c "echo date"; cat echo*

- CVE-2014-7186: Esta nueva variante está provocada por las mismas vulnerabilidades que hemos comentado anteriormente, pero aprovecha el uso de múltiples declaraciones de "**<<EOF**":

*bash -c 'true <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF <<EOF' || echo "CVE-2014-7186 vulnerable, redir_stack"*

- CVE-2014-7187: Se trata de otra variante, utilizando diferentes declaraciones de "**done**" para la ejecución de código:

*(for x in {1..200};*
*do echo "for x$x in ; do :";*
*done;*
*for x in {1..200};*
*do echo done;*
*done)*
*| bash || echo "CVE-2014-7187 vulnerable, word_lineno"*

# EXPLICACIÓN TÉCNICA

- La primera vulnerabilidad aprovecha como las definiciones de funciones se exportan codificadas dentro de la lista de variables de entorno, cuyos valores comienzan con unos paréntesis ("()"), seguido de la definición de la función. La nueva instancia de Bash, al inicializarse, explora el contenido de estas variables de entorno en búsqueda de valores que correspondan a este formato, y los convierte en funciones internas.

- Analizaremos el siguiente fragmento de código, encargado de importar las variables de la función, correspondiente al fichero "**variables.c**":

```
/* Initialize the shell variables from the current environment.
   If PRIVMODE is nonzero, don't import functions from ENV or
   parse $SHELLOPTS. */
void initialize_shell_variables (env, privmode)
     char **env;
     int privmode;
{
  [...]
  for (string_index = 0; string = env[string_index++]; )
    {

      [...]
      /* If exported function, define it now.  Don't import functions from
     the environment in privileged mode. */
      if (privmode == 0 && read_but_dont_execute == 0 && STREQN ("() {", string, 4))
      {
        [...]
        parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
        [...]
      }
}
```

- Observamos un bucle sobre todas las variables de entorno proporcionadas a la función y después un sentencia "**if**" para realizar la comprobación de si nos encontramos en modo privilegiado, que se encontrará desactivado en la mayoría de los casos.

- Aquí es donde ocurre nuestra afirmación "**pero no verifica que ese fragmento es simplemente la definición de una función**", dentro de la función "**parse_and_execute**", cuya descripción y código se encuentra dentro de "**builitins/evalstring.c**":

```
/* Parse and execute the commands in STRING.  Returns whatever
   execute_command () returns.  This frees STRING.  FLAGS is a
   flags word; look in common.h for the possible values.  Actions
   are:
    (flags & SEVAL_NONINT) -> interactive = 0;
    (flags & SEVAL_INTERACT) -> interactive = 1;
    (flags & SEVAL_NOHIST) -> call bash_history_disable ()
    (flags & SEVAL_NOFREE) -> don't free STRING when finished
    (flags & SEVAL_RESETLINE) -> reset line_number to 1
*/
int parse_and_execute (string, from_file, flags)
     char *string;
     const char *from_file;
     int flags;
{
```

- Como observamos en el código, todo lo que se le ha pasado a la función es ejecutado como si fuera un comando ordinario de Bash. Los flags **SEVAL_NONINT** y **SEVAL_NOHIST**, que corresponden a las características de una shell interactiva, así como a la incorporación de historial a Bash, no impiden incluir otros elementos diferentes a las definiciones de funciones.

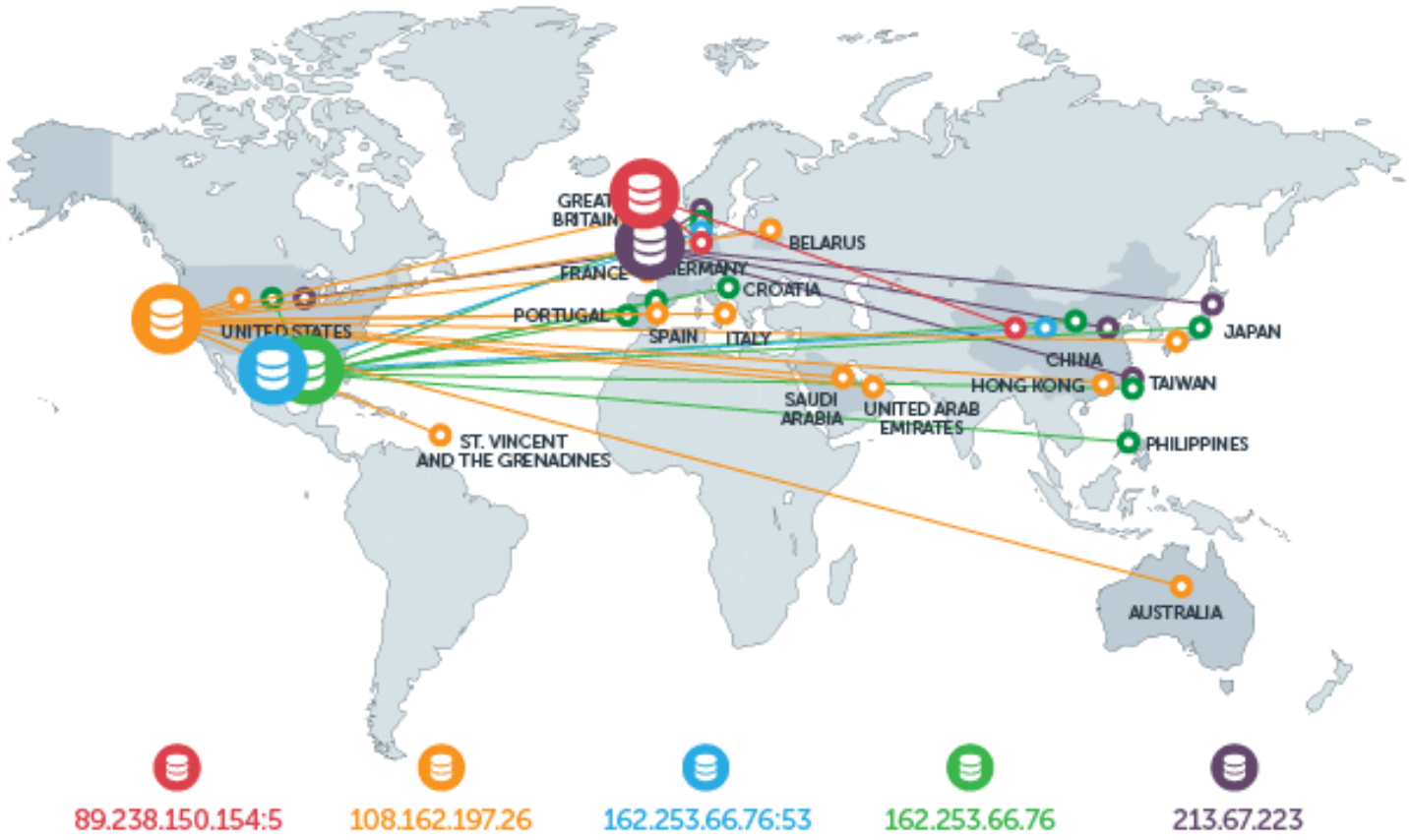
- El parche desarrollado por el equipo de Bash introduce nuevos flags, como **SEVAL_FUNCDEF** y **SEVAL_ONECMD** que se pueden pasar en el campo de flags de la función "**parse_and_execute**"

```
+ #define SEVAL_FUNCDEF 0x080      /* only allow function definitions */
+ #define SEVAL_ONECMD  0x100      /* only allow a single command */
```

- El parche también añade funcionalidades a "**parse_and_execute**" para cumplir con las modificaciones y los flags añadidos:

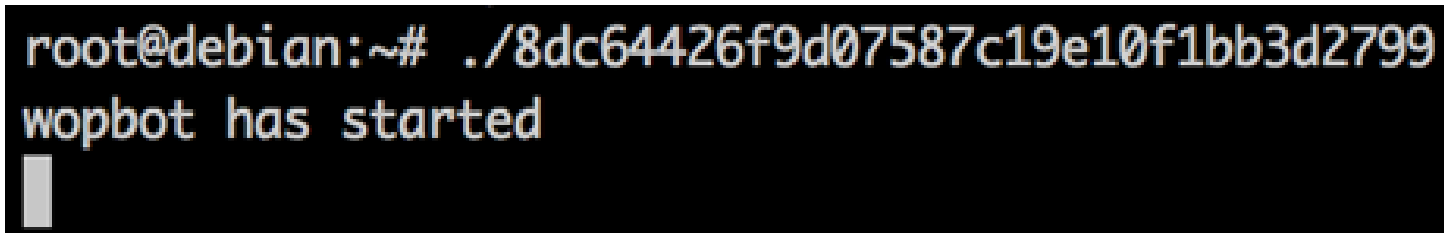
```
-     parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST);
+     /* Don't import function names that are invalid identifiers from the
+        environment. */
+     if (legal_identifier (name))
+       parse_and_execute (temp_string, name, SEVAL_NONINT|SEVAL_NOHIST|SEVAL_FUNCDEF|SEVAL_ONECMD);
```

# TÉCNICAS DE EXPLOTACIÓN 'IN THE WILD'



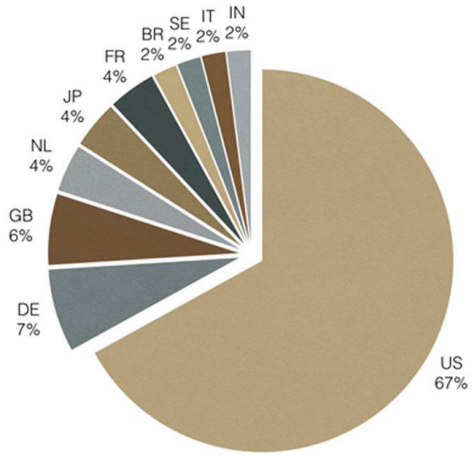

- El día 25 de Septiembre, se reportaron diferentes ataques sufridos por corporaciones, que se focalizaban en ataques de DDOS a través de paneles C&C específicos, utilizando la vulnerabilidad de Bash y diferentes payloads para el compromiso.

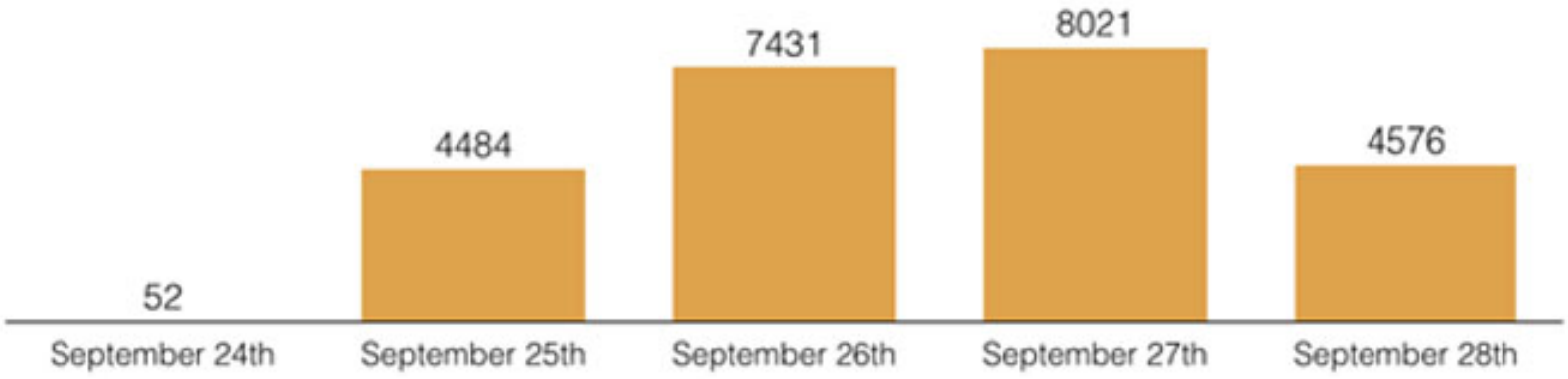
- El 26 de Septiembre, se conoció también la existencia de una botnet que utilizaba diferentes equipos comprometidos, denominada "wopbot". Esta botnet estaba siendo utilizada para realizar diversos ataques DDOS contra Akamai Technologies y para realizar diversos escaneos del Departamento de Defensa de los Estados Unidos:




- Los sucesivos análisis del binario dentro de un entorno controlado mostraron que los diferentes escaneos realizados sobre la red del DoD tenían como fin descubrir servicios de telnet activos en las diferentes plataformas escaneadas, así como realizar ataques de fuerza bruta para conseguir acceso.

- En las siguientes semanas y meses de explotación de la vulnerabilidad, se han encontrado diferentes técnicas y payloads enfocadas al:
  - Fraude publicitario
  - Obtención de shell inversa
  - Robo del fichero de contraseñas del sistema
  - Reconocimiento remoto de direcciones de correo
  - Instalación de backdoors
  - Creación de botnets y clientes DDOS




try it yourself
DEMO

# CONCLUSIONES

## LINUX

- Red Hat, Fedora, CentOS o Yellow Dog Linux:

  *# yum update bash -y*

- Para sistemas Ubuntu o Debian:

  *# apt-get update; apt-get install --only-upgrade bash*

- Para Arch Linux:

  *# pacman -Sy bash*

## OSX

- Para sistemas corriendo OSX, Apple ha publicado parches correspondientes a las siguientes versiones de su sistema operativo que se encuentran en soporte aún:
  - Mavericks: *http://support.apple.com/kb/DL1769?viewlocale=en_US&locale=en_US*
  - Mountain Lion: *http://support.apple.com/kb/DL1768*
  - Lion: *http://support.apple.com/kb/DL1767*

- Si no deseamos utilizar los parches oficiales, también podremos descargar y compilar Bash nosotros mismos utilizando brew o MacPorts.

- **E-Mails**
  - ccn-cert@cni.es
  - info@ccn-cert.cni.es
  - ccn@cni.es
  - sondas@ccn-cert.cni.es
  - redsara@ccn-cert.cni.es
  - carmen@ccn-cert.cni.es
  - organismo.certificacion@cni.es
- **Websites**
  - www.ccn.cni.es
  - www.ccn-cert.cni.es
  - www.oc.ccn.cni.es

Síguenos en Linked in