Evan Parton
Final Project -- COMP 116
Transitioning to a Connected Age of Things
December 14, 2015

For Supporting Material, I chose to analyze source code for wemo.py. WeMo is a switch that allows for connection of any electrical device to an outlet, and remote control of allowing or disallowing the connection of this electrical device. This python script uses a UPnP (Universal Plug and Play) library called Miranda, which allows for generic hooks into the UPnP device. We see in wemo.py an efficient use of the Miranda UPnP library's functions for setting up a connection to the device. A section of the Miranda is included to demonstrate some of the functionality of the *upnp* class object used in wemo.py.

The idea behind wemo.py is demonstrated in tail with the *on()* and *off()* functions, which allow control of the WeMo device. First let us examine the setup required to use these principal functions. First and foremost we set up an instance of the upnp object called *conn*, which offers us the rich ability of Miranda's functions into the UPnP device.

The function *_send()* sets up the relevant host information to connect to the WeMo device, and sends it via a SOAP request (a messaging protocol that allows cross-platform communication via HTTP and XML) to the device. The return value of *_send()* is that of the return value of *sendSOAP()*, a function from Miranda's *upnp* class, which is 'False' upon failure or the body of the SOAP request upon success. We use this body response to extract a success state (tag 'Error').

To enable the functionality of *on()* and *off()*, we first need to run the discovery function *get()*, which simply finds the current state of the WeMo device. Following, we may turn on the electronic device attached to WeMo with *on()*, and inversely off with *off()*. These functions work by using the *send()* function to communicate to the WeMo device a change of BinaryState (i.e. 'on' and 'off'). It does a final check to confirm the return is not an error (in which case returns 'False' to the prompt), and otherwise returns 'True' to the prompt. For further details for the potential danger of this exploit, please see main paper.

wemo.py:

```
==========================================
from miranda import upnp, msearch

conn = upnp()
msearch(0, 0, conn, 2)

SWITCHES = []

# populate all the host info, for every upnp device on the network
for index in conn.ENUM_HOSTS:
    hostInfo = conn.ENUM_HOSTS[index]
    if hostInfo['dataComplete'] == False:
        xmlHeaders, xmlData = conn.getXML(hostInfo['xmlFile'])
        conn.getHostInfo(xmlData,xmlHeaders,index)


for index in conn.ENUM_HOSTS:
    try:
        if conn.ENUM_HOSTS[index]['deviceList']['controllee']['modelName'] == 'Socket':
            SWITCHES = [index]
    except KeyError:
        pass


def _send(action, args=None):
    if not args:
        args = {}
    host_info = conn.ENUM_HOSTS[SWITCHES[0]]
    device_name = 'controllee'
    service_name = 'basicevent'
    controlURL = host_info['proto'] + host_info['name']
    controlURL2 = hostInfo['deviceList'][device_name]['services'][service_name]['controlURL']
    if not controlURL.endswith('/') and not controlURL2.startswith('/'):
        controlURL += '/'
    controlURL += controlURL2

    resp = conn.sendSOAP(
        host_info['name'],
        'urn:Belkin:service:basicevent:1',
        controlURL,
        action,
        args
```

```
    )
    return resp

def get():
    """
    Gets the value of the first switch that it finds
    """
    resp = _send('GetBinaryState')
    tagValue = conn.extractSingleTag(resp, 'BinaryState')
    return True if tagValue == '1' else False

def on():
    """
    Turns on the first switch that it finds.
    BinaryState is set to 'Error' in the case that it was already on.
    """
    resp = _send('SetBinaryState', {'BinaryState': (1, 'Boolean')})
    tagValue = conn.extractSingleTag(resp, 'BinaryState')
    return True if tagValue in ['1', 'Error'] else False

def off():
    """
    Turns off the first switch that it finds.
    BinaryState is set to 'Error' in the case that it was already off.
    """
    resp = _send('SetBinaryState', {'BinaryState': (0, 'Boolean')})
    tagValue = conn.extractSingleTag(resp, 'BinaryState')
    return True if tagValue in ['0', 'Error'] else False
==============================================
source: https://github.com/issackelly/wemo/blob/master/wemo.py

miranda.py:
==============================================
#UPNP class for getting, sending and parsing SSDP/SOAP XML data (among other things...)
class upnp:
        ip = False
        port = False
        completer = False
        msearchHeaders = {
                'MAN' : '"ssdp:discover"',
                'MX'  : '2'
        }
        DEFAULT_IP = "239.255.255.250"
```

```
DEFAULT_PORT = 1900
UPNP_VERSION = '1.0'
MAX_RECV = 8192
HTTP_HEADERS = []
ENUM_HOSTS = {}
VERBOSE = False
UNIQ = False
DEBUG = False
LOG_FILE = False
IFACE = None
STARS = '**********************************************************************'
csock = False
ssock = False

def __init__(self, ip=False, port=False, iface=None, appCommands=[]):
        if appCommands:
                self.completer = cmdCompleter(appCommands)
        if self.initSockets(ip, port, iface) == False:
                print 'UPNP class initialization failed!'
                print 'Bye!'
                sys.exit(1)
        else:
                self.soapEnd = re.compile('<\/.*:envelope>')

#Send network data
def send(self,data,socket):
        #By default, use the client socket that's part of this class
        if socket == False:
                socket = self.csock
        try:
                socket.sendto(data,(self.ip,self.port))
                return True
        except Exception, e:
                print "SendTo method failed for %s:%d : %s" % (self.ip,self.port,e)
                return False

#Listen for network data
def listen(self,size,socket):
        if socket == False:
                socket = self.ssock

        try:
                return socket.recv(size)
```

```
                except:
                        return False

        #Create new UDP socket on ip, bound to port
        def createNewListener(self,ip=gethostbyname(gethostname()),port=1900):
                try:
                        newsock = socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP)
                        newsock.setsockopt(SOL_SOCKET,SO_REUSEADDR,1)
                        newsock.bind((ip,port))
                        return newsock
                except:
                        return False

        #Return the class's primary server socket
        def listener(self):
                return self.ssock

        #Return the class's primary client socket
        def sender(self):
                return self.csock

        #Parse a URL, return the host and the page
        def parseURL(self,url):
                delim = '://'
                host = False
                page = False

                #Split the host and page
                try:
                        (host,page) = url.split(delim)[1].split('/',1)
                        page = '/' + page
                except:
                        #If '://' is not in the url, then it's not a full URL, so assume that it's just a
relative path
                        page = url

                return (host,page)

        #Send GET request for a UPNP XML file
        def getXML(self, url):

                headers = {
                        'USER-AGENT':'uPNP/'+self.UPNP_VERSION,
```

```
                'CONTENT-TYPE':'text/xml; charset="utf-8"'
        }

        try:
                #Use urllib2 for the request, it's awesome
                req = urllib2.Request(url, None, headers)
                response = urllib2.urlopen(req)
                output = response.read()
                headers = response.info()
                return (headers,output)
        except Exception, e:
                print "Request for '%s' failed: %s" % (url,e)
                return (False,False)

    #Send SOAP request
    def sendSOAP(self, hostName, serviceType, controlURL, actionName,
actionArguments):
        argList = ''
        soapResponse = ''

        if '://' in controlURL:
                urlArray = controlURL.split('/',3)
                if len(urlArray) < 4:
                        controlURL = '/'
                else:
                        controlURL = '/' + urlArray[3]


        soapRequest = 'POST %s HTTP/1.1\r\n' % controlURL

        #Check if a port number was specified in the host name; default is port 80
        if ':' in hostName:
                hostNameArray = hostName.split(':')
                host = hostNameArray[0]
                try:
                        port = int(hostNameArray[1])
                except:
                        print 'Invalid port specified for host connection:',hostName[1]
                        return False
        else:
                host = hostName
                port = 80
```

```
		#Create a string containing all of the SOAP action's arguments and values
		for arg,(val,dt) in actionArguments.iteritems():
			argList += '<%s>%s</%s>' % (arg,val,arg)

		#Create the SOAP request
		soapBody =	"""<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
 <s:Body>
  <u:%s xmlns:u="%s">
   %s
  </u:%s>
 </s:Body>
</s:Envelope>
""" % (actionName, serviceType, argList, actionName)

		#Specify the headers to send with the request
		headers =	{
			'Content-Type':'text/xml; charset="utf-8"',
			'SOAPACTION':'"%s#%s"' % (serviceType,actionName),
			'Content-Length': len(soapBody),
			'HOST':hostName,
			'User-Agent': 'CyberGarage-HTTP/1.0',
		}

		#Generate the final payload
		for head,value in headers.iteritems():
			soapRequest += '%s: %s\r\n' % (head,value)
		soapRequest += '\r\n%s' % soapBody

		#Send data and go into recieve loop
		try:
			sock = socket(AF_INET,SOCK_STREAM)
			sock.connect((host,port))
			sock.send(soapRequest)
			while True:
				data = sock.recv(self.MAX_RECV)
				if not data:
					break
				else:
					soapResponse += data
					if self.soapEnd.search(soapResponse.lower()) != None:
						break
```

```
                        sock.close()

                        (header,body) = soapResponse.split('\r\n\r\n',1)
                        if not header.upper().startswith('HTTP/1.1 200'):
                                print 'SOAP request failed with error
code:',header.split('\r\n')[0].split(' ',1)[1]
                                errorMsg = self.extractSingleTag(body,'errorDescription')
                                if errorMsg:
                                        print 'SOAP error message:',errorMsg
                                return False
                        else:
                                return body
                except Exception, e:
                        print 'Caught socket exception:',e
                        sock.close()
                        return False
                except KeyboardInterrupt:
                        sock.close()
                return False
```

==========

source: https://github.com/issackelly/wemo/blob/master/miranda.py