Survey on Server Side Request Forgery

(An Overview of SSRF)

Janaki Reddy Gaddam
School of Informatics, Computing and Engineering
Indiana University
Bloomington, Indnana, U.S.A
jgaddam@iu.edu

Nawaz Hussain Khazielakha
School of Informatics, Computing and Engineering
Indiana University
Bloomington, Indiana, U.S.A
nawazkh@iu.edu

***Abstract*—Server Side Request Forgery is an evolving vulnerability in today's web application, where an attacker sends malicious requests to a server in an attempt to access its internal network configuration and resources. Attacker may use a combination of content based, error based, bool based, and time based server side request forgery methods to perform intranet scan, probe supported protocols and request internal (configuration or password) files.**

Using this data, attacker may perform additional operations such as URL redirection, SQL injection, remote file inclusion and cross site scripting (XSS). Prevention techniques such as network level-egress filtering, url validation, origin authentication and disabling unused ports work partially to counter SSRF attacks. This paper focuses on providing a comprehensive overview on server side request forgery types and attacks, providing its history and real world examples.

This Paper also proposes defense mechanisms to defend against SSRF attacks and also discusses its similarity with Cross Site Request Forgery attack.

***Index Terms*—SSRF, web vulnerability, Intranet probing, SSRF port scan, CSRF, URL validation**

I. Introduction

Today's web-applications architecture range from a monolithic build to a completely modularized distributed systems which are highly available and scalable. Irrespective of their build architecture, web-application's back-end, known as servers, are meant to handle/resolve user requests. Each request is usually sent to a central routing/distributing server which forwards the extracted request to an appropriate internal server to resolve the user request. These internal servers run on a secure intranet, and all the direct communication to these functional servers are prohibited by design and can only be reached through the source server as other servers only provide functionality support to the application.

In a typical request, the source server forwards a requests to an internal server running. These inter-server requests are at times user originated and at times are application generated. Such intra-network inter-server communication to resolve a user query helps in lowering the processing load from an application, improves turn around time and also enables the developers to trace specific issue if it arises.

Although server-side requests help web-applications in multiple ways, it also poses a major threat in real-time web systems. Since these server-side requests bypass firewalls to access internal servers, attackers try to take advantage of this feature to target a system with SSRF vulnerability. Attackers craft malicious requests which are sent to a source server of a web application . These requests do not appear malicious on first glance, however attackers main motto is to gain knowledge on web-applications internal resources using such requests. SSRF vulnerability can arise due to framework loop holes or careless programming practices used in the implementation of the web application.

Server Side Request Forgery is an attack concept, first presented in 2008. Quoting the Common Weakness Enumeration (ID: 918) description for SSRF, In an SSRF attack

> *" The web server receives a URL or similar request from an upstream component and retrieves the contents of this URL, but it does not sufficiently ensure that the request is being sent to the expected destination."*

To explain SSRF in a simpler terms, imagine you have a worker who helps you carry goods from the factory. If the worker brings *'every'* requested good, without checking if the requested item belongs to you or not, you can exploit this flaw to request other (users) goods. Servers with SSRF vulnerability also act like a careless worker.

Attackers often use the source servers as a proxy to attack the internal resources. Hacker use them to bypass firewall and perform multiple actions such as intranet/port scan, probe supported protocols and retrieve configuration files to draft a detailed attack plan and find loopholes. Using this data, attacker may perform additional operations such as URL redirection, protocol smuggling, local file extraction and cross site scripting (XSS) [2]. SSRF is a serious vulnerability which is often neglected by security researchers and developers as exploitation depends on vulnerability in the intranet than in the application itself. We have cataloged a few recent notable SSRF attacks in this paper to emphasize this vulnerability's significance.

In our research we found that the web development frameworks built on PHP, Java were also patched and vulnerable internal functions were deprecated to mitigate this vulnerability. Current prevention techniques like URL validation, origin

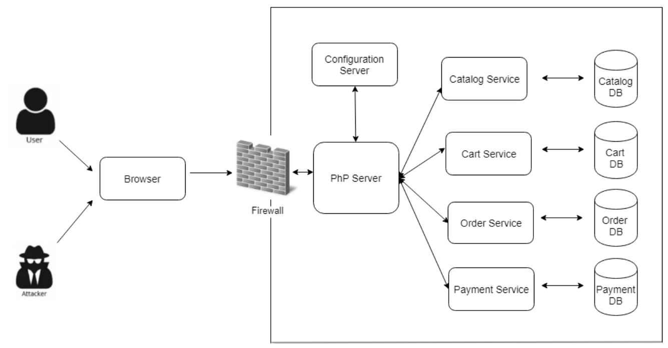

Fig. 1. Sample architecture of a web application with php server as the source server and other services running in a secure intranet which is protected by a firewall. Direct connection from the internet to any other service apart from php server is blocked in this architecture.

authentication, blocking of unused ports and disabling legacy protocols provide a certain level of protection against SSRF but are not a complete solution.

SSRF vulnerabilities can be divided into five categories, Content based, bool based, error based, blind based, and time based SSRF based on the response received from target server for the malicious request. Using the content based, error based and bool based exploitation, the attacker can directly know if the request reached to the intended URL and hence they are also grouped under direct damage category. The other two categories of attack are hard to draw conclusions from and need deeper understanding of the web-application under attack to spot the loop holes, and hence they are grouped under indirect damage type.

We observed that SSRF is often confused as Cross Site Request Forgery (CSRF) in a normal discussion. SSRF vulnerability lets the attacker learn more about the internal network (intranet), where as in CSRF, attacker does not focus on attacking the internal resources of source application and rather exploits the trust factor between two web-applications. Attacker attempts to trick the user in sending a malicious request to another application using the identification of the victim user.

II. Evolution of SSRF

Server Side Request Forgery was first discussed by Deral Heiland at Shmoocon conference in 2008. The topic of discussion was Web Portals Gateway to Information or A Hole In Our Perimeter Defences [3]. In this conference, discussion was around how web-interface allows loading files from other HTTP sources and this was shown by web portals that were designed to deliver the user requested information that user cannot access directly. These portals make a single point of access to internal resources to retrieve the requested data.

Over the years, attackers have used different techniques to exploit SSRF vulnerability in a web application. SSRF attacks initially started with overwriting files such as autologon.bat with OS commands to compromise Oracle database server. In this attack, a Transparent Network Substrate (TNS) [1] packet was created to execute a remote function called set_log_file to bypass local authentication restriction on Oracle 10G databases. A TNS (Transparent Network Substrate) name is the name of the entry in tnsnames.ora file which is kept in $ORACLE_HOME/network/admin [4]

Another old way of exploiting SSRF is to use SMBRelay(Server Message Block Relay), a way to exploit universal naming convention (UNC) path having minimum rights. SMBRelay is done through many ways like from MySQL, from Oracle DB, from browsers, from USB, by IP spoofing, from SAP NetWeaver ABAP, from SAP NetWeaver J2EE.

Current methods to exploit SSRF are due to the vulnerabilities found in different underlying components of a web application like

- Format processing component - XML (DTD remote access, XML design), OpenOffice (Dynamic data linking, External resource embedding).
- Direct sockets access - CRLF injection: Apache webserver with LF splitter is vulnerable to SSRF if attacker transmits a packet with 0x0d byte ,and Nginx web-server HTTP parser is also vulnerable as the server supports splitters without CR byte (0x0d).
- Net library URL processing (unsafe server side redirect and others) - cURL (cURL is a command-line tool for getting or sending files using URL syntax), Library for WWW in Perl (LWP)
- File Descriptors Exploitation-Useful in cloud, shared hosting and large infrastructures. FDE can be done by Interpreters API, exec() call from API, ProCFS files.
- Bypassing restrictions Include bypassing input validation by unsafe redirects, domain name server (DNS) pinning and network restrictions on firewalls.

III. SSRF Classifications

Assume that a web server is already identified with an SSRF vulnerability, then an attacker can perform five types of attacks. These attacks have been categorized based on the response received from the targeted server.

To explain the types in more detail, consider the below php script which handles user request to process an uploaded image (.png) file. Uploaded file is sent as a parameter in the url to this php file(name testSSRF.php).

$<?php$
$if(isset(\$\_GET[url])){$
$\$url = \$\_GET[url];$
$\$content = file\_get\_contents(\$url);$
$header(Content - Type : image/png);$
$Echo\$content;$
$\# - - - - - - - - - - - - -$

$\#codetoperformactionontheuploadedimage$

$\# - - - - - - - - - - - - -$

$header(HTTP/1.1301MovedPermanently);$
$header(Location : /gohere.php);$

```
}else{
Echo "set url to change your picture"
}
?>
```

A. Content Based

If the server responds back to the malicious request by replying back with the requested content, then it is called as content based SSRF. This is one of the most dangerous vulnerability which can exist in a web application, and developers/designers should take utmost care in handling such requests. To the above code snippet, the expected request would look like:

http://<myDomainName>/testSSRF.php?url=myImage.png

However, a attacker use the same request to extract internal password/configuration file by using the following request

http://<myDomainName>/testSSRF.php?url=file:///etc/passwd

The server would respond with the requested file (if present locally) because of the flawed implementation of file_get_contents function of php framework. Moreover, all the contents of the data are available at the console of the source server because of the *'echo $content'* command. And therefore this is the most dangerous SSRF vulnerability.

B. Bool Based

In this SSRF attack, the server responds with a null or false if the resource does not exists at the specified URL. This type of SSRF attack is closely related with Time based SSRF attack. Using the response of the server, the attacker can change the malicious request to target another internal IP.

For the server hosting above php file, if the server returns a null or 'unable to reach it status' any output on consuming the following request:

http://<myDomainName>/testSSRF.php?http://10.0.0.1/

It means that there is no service running on the ip 10.0.0.1.

C. Error Based

The server responds with http statues to the SSRF attacks. If such a response is sent by the server, it implies that there is server running at that IP address, but the port or the request parameters do not match with a working service URL. However after multiple trial and error attacks, attacker may be able to crack the right url to access the service.

For the server hosting above php file, if the server returns http error status (404 or 405) on consuming the following request :

http://<myDomainName>/testSSRF.php?http://10.0.0.1/get/data

It means that there is a service active on the mentioned IP, however the request parameters are not correct.

D. Time Based

Servers often take a lot of time to respond when a service requested is not active on registered IP. This is because of the time delay due to multiple connection establishment trials before the server timeouts. Attacker can use the delayed bool response to learn whether a service exists in the intranet or not.

E. Blind Based

From the attackers perspective, this is one of the toughest to crack SSRF vulnerability as the attacker will not know if the requested IP exists in the secure intranet. This SSRF is similar to time based SSRF but the response for a malicious request is always replied after a fixed time interval.

Above types of SSRF vulnerabilities can again be grouped into two broad categories **direct SSRF** and **indirect SSRF**, based on the attacker's prior knowledge about the target server (or IP address). Bool based, error based, and content based SSRF are grouped under direct SSRF as the service to be attacked is known to the attacker. Time based and blind based SSRF attacks are grouped in indirect SSRF category as the attacks are targeted to the services whose existence is not known prior to the attack.

IV. SSRF Attack Approaches

An attackers ultimate goal is to use the above mentioned SSRF attacks to learn all the living IP addresses, supported protocols and identify unauthenticated services running in the internal network open for exploitation. But for an exploitation to occur, attacker has to find out if the web application contains an SSRF vulnerability. Attacker initially tries to search for ports which are open and check for supported protocols. Skanda [5], an SSRF exploitation tool provided by OWASP can perform port scanning on a web-application to find SSRF vulnerabilities. However, one application cannot reveal everything about the target, hence the attacker will develop an attack plan based on the responses received from basic probing. Then using the data from basic probe, attacker can use bool based, time based, content based attacks to retrieve more data.

A. Basic Probing or Intranet Scanning

In this approach, the agenda of the attacks is to identify if the server has an SSRF vulnerability and not to classify it into a category. Attacker uses HTTP GET or POST request to hit the target web-server and monitor its responses. Usually, multiple hosts are configured to send malicious requests to the target and the responses are monitored for hints.

The internal IPs range from 10.0.0.0 - 10.255.255.255, 172.16.0.0 - 172.31.255.255 and 192.168.0.0 - 192.168.255.255, in total 17000000 IPs. Since it is highly unlikely to setup internal firewalls between internal services, attacker may hit standard ports of random IPs to check if the target IP is being used in the network.

It is possible that a server with SSRF might only indicate whether a URL is reachable or not, but this is a crucial knowledge as the attacker can change the attack strategy later on to improve success rate. Once sufficient information is gathered from basic probing/intranet scanning, attacker moves on to advanced probing.

B. Advanced Probing

In advanced probing, attackers malicious requests will focus on categorizing the SSRF vulnerability. Attacker will try all the types of SSRF attacks (as mentioned in the types of SSRF) to learn more about the internal network and resources.

In addition to categorizing the SSRF vulnerability, identifying the supported protocols in the intranet is very important in exploiting a target. Attackers can do a lot of damage to the web-application by smuggling protocols into the intranet [6]. Yin Wang provided a list of exploitable protocols:

- file:// - Access local files
- gopher:// -Was used as an alternative prior to world wide web to search documents
- ftp:// - file transfer protocol
- php:// - Access values I/O stream
- http:// - Access HTTP URLs
- dict:// - Dictionary Protocol
- phar:// - PHP Archive

Out of which all of them might not work in the source server due to framework language differences or few of them being filtered by firewall. Classic example of protocol utilization in an SSRF attack is usage of *'file:///etc/passwd* in Content Based SSRF attack where the password file was accessed from the server.

To test whether a protocol is supported or not, attacker will have to look into the response of error based SSRF request. If the response is normal then it is implied that protocol is supported in the intranet.

C. Using the CRLF injection

Another important step in carrying out SSRF attack is to look for Carriage-Return Line-Feed(CRLF) support in the malicious request. Using CRLF injection, the attacker will have greater control on executing custom requests on the source server and can damage the internal network more severely.

Carriage-Return (ASCII 13, '\r') Line-Feed (ASCII 10, '\n') in modern system is used to terminate a line of HTTP request. Using CRLF encoding in a request is similar to pressing 'return' key. CRLF if not filtered at the application level can be exploited by the attacker to execute custom commands. Attacker can use CRLF injection to smuggle different protocols into the intranet. These protocols enable the attacker to transform them to exploitable protocols by adding CRLF encoding and custom parameters.

For example: consider there is a SMTP server without authentication running in the intranet, a SSRF request could be framed as:

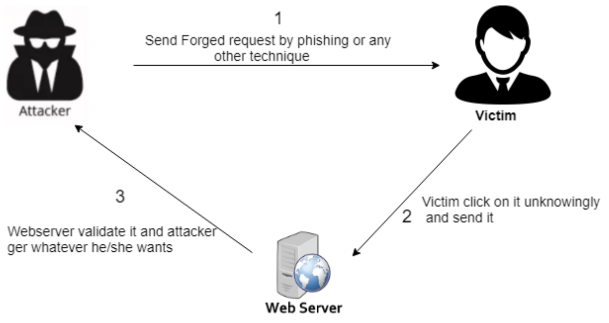
1
Send Forged request by phishing or any other technique
Attacker
Victim
3
Webserver validate it and attacker ger whatever he/she wants
Victim click on it unknowingly and send it
2
Web Server

Fig. 2. Cross Site Request Forgery attack procedure

https://myDomainName%0D%0ADear evil.com%0D%0AMAIL FROM...:25/

The SMTP server in the intranet would receive:

```
Get /
Host:myDomainName:25
Dear Evil.com
Mail from
```

First two lines in the request are invalid, however, the next two lines will make the server send a forged email. If the attacker receives the email, then attacker can confirm that there is SMTP server running in the intranet without any authentication.

CRLF injection is a powerful tool in the attackers arsenal to exploit the internal network. It can be used to set http parameters in the custom URL, set user defined variables and execute text based commands like run docker commands: *'docker stack rmi'* to take down running instances of an application.

V. SSRF Vs CSRF

While SSRF vulnerability allows an attacker to alter a parameter used on the web applications to take full/partial control of the internal systems, these systems are behind firewall that are usually not accessible to an attacker from the external network. Cross-site Request Forgery (CSRF) is web vulnerability where malicious requests are sent from one web application to another. We have presented a key contrasts on the functioning of CSRF when compared to SSRF.

In a CSRF attack, a malicious entity tricks the victim to perform actions on behalf of the attacker. The impact of the attack depends on the level of permissions acquired by the attacker.

CSRF is implemented in two steps:

- Victim is tricked into clicking a malicious link or loading up a web page, this is usually done by using social engineering techniques.
- Next, attacker sends a maliciously crafted request in the victims browser, the crafted request looks like a legitimate

request to the browser. The purpose of the request is to get hold of the sensitive values present in a session like cookies.

To put it in much simpler terms, CSRF takes the advantage of browser that sends cookies to web applications with each and every request.

But the tricky part in CSRF is that, the attack will be effective only if the victim is authenticated. Though CSRF attacks are used to bypass the authentication process, elements like publicly accessible web pages are not affected by CSRF, because a low privilege user cannot perform high privilege actions on the server.

Clickjacking [7] and Login CSRF [8] are other two kinds of CSRF.

VI. Real World Examples of SSRF

A. Critical SSRF Vulnerability in Paypals subsidiary allows to access Internal Network

In 2014, PayPal publicly acknowledged the SSRF vulnerability in one of their servers. The vulnerability was reported by a security researcher - Shubam Shah on merchants.billmelater.com, a part of Paypal infrastructure. The vulnerability was found on a testbed for BillMeLaters SOAP API, which allowed querying to any given host URL.

The API exposed internal URLs, which indirectly give access to internal hosts within BillMeLater/PayPal network. The researcher claimed that an attacker could gain access to customer data by accessing components of the internal network.

B. SSRF flaw exposed information from Googles internal network [9]

In May 2018, security engineer Enguerran Giller discovered SSRF vulnerability on Google Caja server. Google Caja is a compiler tool that is used for securely embedding third party HTML, JavaScript and CSS code into our websites.

SSRF vulnerability was discovered when Giller was trying to reproduce XSS on an unpatched version of Caja. Giller hosted a JavaScript file on Google Cloud services and bypassed the limitation of fetching information from Google servers. Giller was continuously testing the Googles network until Google rolled out a fix.

Though he was not able to achieve unrestricted file access or remote code execution, he could access interesting information from Googles Borg, a datacenter management system that runs the companys services. Google applied the patches in less than 48 hours from the report.

C. Severe Vulnerability exposes Wordpress websites to attack. August 16, 2018

In August 2018, Secarma researcher Sam Thomas told that a wordpress vulnerability had the potential to take down many websites running on their content management system.

Vulnerability was found in the WordPress php frameworks upload file functionality where the attacker could upload malicious thumbnails to trigger pharr:// stream wrapper. Using this, the attacker could trigger unserialization of platform code and load custom code and execute it in the WordPress environment. Other real world examples are WooYun-20150163792, WooYun-2015099070.

VII. Defense Mechanisms

It is clear that there cannot be one simple solution to defend against SSRF attacks as there can be numerous vulnerabilities in the web-application framework and implemented code. Therefore, we have summarised the steps to defend against each type of SSRF attack, however it is not an exhaustive list.

We recommend a user implement traditional security measures to defend against SSRF attacks. Traditional methods include Header checking, Token validation, client side defence using regular expressions, and user access control. Moreover, the application may blacklist suspicious IPs and use regular expressions to validate requested URLs. This will be used as initial security and further security will be built on this step.

Users should not be allowed to use their inputs into the functions that make requests to internal resources. There should be a validation check put on each input received and also suspicious inputs should be logged and reported to the application administrator.

It is highly recommended that default ports for services to be remapped to custom ports. For example, Apache server runs at localhost:8080, this should be remapped to another port number which is less often used/known so that in case of a security breach, the attacker cannot easily guess the port to run error based SSRF attack.

Developers should make sure that their code does not print the contents of the requested file even in the hidden logs, as the attacker may get access to the internal logs. Also, IP level access to internal resources should not be allowed in the requested url. For example, for the applications running in docker swarm, the services are addressed by the service name rather than their IPs. Hence, the attackers can only access the service with the service name and all the IP level communications from the internet will be rejected.

Configuration files should be accessible to a user without an additional authentication to prove the legitimacy of the user. Moreover, the admin should be notified on such an access. All the configuration and environment variables should not be kept into one file and should be distributed in the servers. Although, this would mean that the new developers debugging the application will have a hard time in retrieving the values, but this will provide security against SSRF attacks.

Not all the protocols should be accessible to the users. That is, if the attackers forms a malicious request with a protocol like ‘pharr‘in the requested url, this should be filtered at the source server and the users IP should be blacklisted.

There should be a through check on usage of CRLF encodings in the input urls. CRLF injection is a fatal vulnerability as the attacker can smuggle many exploitable protocols and this vulnerability should be addressed in the design of the application and tested in the testing phase of the application.

Ben S. Y. Fung and Patrick P. C. Lee have detailed ongoing work on defending attacks against SSRF and CSRF vulnerabilities. [10]

VIII. Conclusion and Future Work

We have presented a survey of SSRF vulnerabilities and attacks related to it. It is evident that SSRF vulnerabilities are easy to neglect at the time of system design and build. Attacker can take advantage of different types of responses received from the target server and hence it is very important to scrutinize the responses.

One single approach to secure a web application is not sufficient and hence defending SSRF attacks is an incremental process where the process to patch an application begins from the initial development testing phase. Multiple approaches are discussed in this paper to defend against SSRF attacks. These approaches are meant to provide a starting point to the reader but are not complete and an exhaustive list to watch for.

References

[1] https://stackoverflow.com/questions/15819433/difference-between-using-a-tns-name-and-a-service-name-in-a-jdbc-connection

[2] https://slideplayer.com/slide/8924005/

[3] ref: https://slideplayer.com/slide/8924005/

[4] https://stackoverflow.com/questions/15819433/difference-between-using-a-tns-name-and-a-service-name-in-a-jdbc-connection

[5] https://www.owasp.org/index.php/OWASP_Skanda_SSRF_Exploitation_Framework

[6] Wang Y. How to Construct Your SSRF Exploit Framework, page 36. Wooyun Summit 2016.

[7] R. Hansen and J. Grossman. Clickjacking. http://www.sectheory.com/clickjacking.htm, 2008

[8] A. Barth, C. Jackson, and J. C. Mitchell. Robust Defenses for Cross-Site Request Forgery. In Proc. of ACM CCS, 2008.

[9] Google Caja, ref: https://developers.google.com/caja/

[10] https://www.cse.cuhk.edu.hk/ pclee/www/pubs/trustcom11.pdf