Chapter 25
Server Side Request Forgery (SSRF)

Revision 1.2, April 2026.

Objectives

- Understand the causes of server side request forgery attacks.
- Learn about the risks associated with server side request forgery.
- Learn about the damage that can result from such attacks.
- See real world examples of SSRF attacks
- Learn ways to prevent these forgery attacks.

25.1 Motivation

In Chapter 22 on Cross Site Request Forgery attacks, we learned how an attacker could convince a web client to send out a web request that they did not intend. By visiting a website with malicious content, the attacker delivers a web page to an unsuspecting client that contains HTML or JavaScript that causes an unintended web request to be generated by that client.

In this chapter, we will see how a web server might be similarly convinced to send out such an unintended web or other type of request. When a web server sends out an unintended request, it might be directed at internal servers or data sources that only the server can access. Since the server is issuing the request, the server might automatically authenticate itself while making the request. The result is that the attacking client might indirectly initiate actions for which it is not authorized or get access to data (including keys and credentials) for which it is not authorized.

As with injection attacks (Chapter 15-19), including XSS (Chapter 21) and directory traversal attacks (Chapter 12), these attacks are enabled, in part, by lack of checking or sanitization of the input that comes from the client.

25.2 An Example SSRF Attack

We start by thinking about why a server would generate a request to a user-supplied URL. A common feature in a server is the ability of the client to provide the server with a URL or input that will be used to construct a URL. This URL might be for a file to be uploaded or a web page to be checked or reviewed.

As an example, suppose that an attacker is using an interface that is intended to upload documents to their account. When the attacker makes a request, it provides the server with a URL to an internal interface in the server's environment, where this URL points to a store of security credentials.

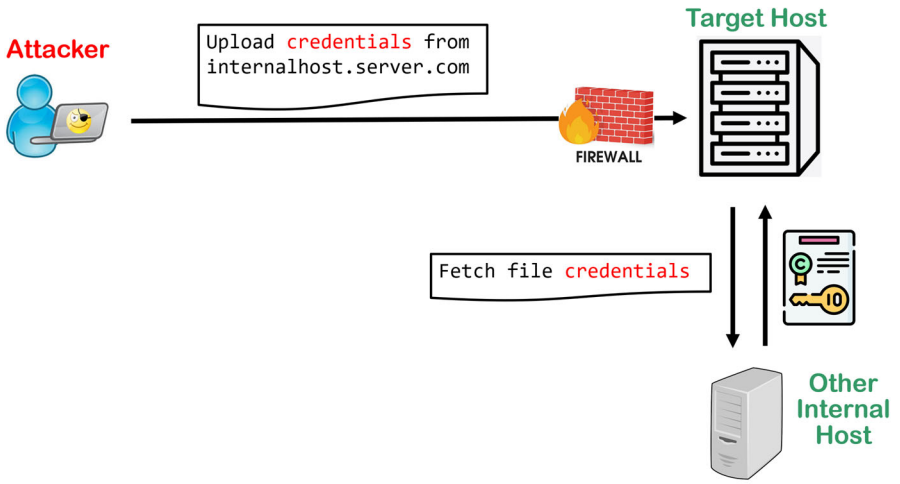

Figure 1: Flow in Typical Server Side Request Forgery Attack

In Figure 1, we see an illustration of the flow of this type of attack. It starts with the attacker, acting as a normal user, asking the server to upload a file to the user's account. However, the attacker points the server at an internal file on a host that would not normally be accessible by a user. The server neglects to check the host name in the request and fetches the file and stores it in the user's account. At some point later, the user downloads that file and make illicit use of the credentials.

The request by the attacker might be directed at a host that the server normally accesses. Or the attack might be directed at a function in the API that should not be allowed by an outside user. Or the attack might be directed at files that should not be accessible by an outside user. So, the checks made by the server might also need to include the function and files being requested.

An important feature that makes this type of attack so profitable is that the attacker convinces the server to make the attack. Having the server make the request can defeat security protections in a couple of ways. First, the server is likely behind a firewall and maybe further protected by network segmentation (where the network routers restrict which hosts can talk with which other hosts). As a result, internal services accessed by the targeted host may not be protected with authentication because they are not reachable by hosts outside of the server's domain. For example, there might be an internal MongoDB database server containing security and configuration data that has its authentication turned off. Or the attack might be directed at the server itself by requesting access to `localhost` or `127.0.0.0`.

Second, even if access to internal hosts does require authentication, it might automatically be generated when the target server makes requests to internal or external servers.

25.3 Two Notable SSRF Attacks

Since SSRF (CWE-918) is in the top 25 of CWEs found in the wild, you know that this type of attack is happening frequently. These attacks have often be quite serious and the number of such attacks has increased steadily each year over the past decade.

Now, we will discuss a couple of significant SSRF attacks. For each one …

Capital One

In 2019, the Capital One financial organization suffered a major breach of security where an attacker obtained the personal and financial information of over 100 million users[1]. This attack took place on services deployed in the AWS EC2 cloud.

A misconfigured Modsecurity web application firewall (WAF)[2] and insufficient intrusion detection system (IDS) rules allowed an attacker to go through the firewall and have direct access to services behind the firewall. The services behind the firewall were not expecting outside connections, so were not sufficiently protected against unexpected and unauthorized request. This is an example of lack of defense in depth.

The attacker sent a request to the improperly protected service that caused it to generate a request to the AWS Metadata Service running in that cloud partition. The URL that the attacker caused to be generated looked something like:

```
http://169.254.169.254/latest/meta-data/iam/security-
credentials/ISRM-WAF-Role
```

This request used a local host IP address – i.e., one that is only valid within the firewall space, not from the outside – to request security credentials. The IP address was of the form `169.254.X.X`. An address of this form (called APIPA, Automatic Private IP Addressing), is normally assigned automatically by Windows when a DHCP server is unavailable. The use of this type address in the request was a clear sign that something strange was happening, and this strangeness should have triggered a response from the IDS.

The requested security credentials were later used to access and control resources that furthered the attack. There were several things that allowed this attack to happen. First, the WAF was configured incorrectly. Second,

---

[1] S. Khan, I. Kabanov, Y. Hua, and S. Madnicks, "A Systematic Analysis of the Capital One Data Breach: Critical Lessons Learned", *ACM Transactions on Privacy and Security,* **26**, 1, November 2022, `https://doi.org/10.1145/3546068`.

[2] `https://modsecurity.org/`

the attacked server should not have accepted a connection from outside. Third, the attacked server should never have forwarded the request (the actually SSRF part of the attack). And last, the IDS should have detected these actions.

Microsoft Exchange

A few years ago, analysts and attackers started to look at the Microsoft Exchange mail service. The attacks were based on the fact that Exchange has a frontend server that receives requests and then routes them to a backend. If an attacker structured a request properly, it could convince Exchange to route it to an inappropriate place, resulting in such exploits as remote code execution or file overwrites. Such a major attack, reported as CVE-2021-26855[3], originated from a state-sponsored advanced persistent threat (APT) group known as HAFNIUM[4].

Microsoft Exchange Client Access Service (CAS) is the frontend that accepts all incoming requests, no matter what protocol is used and then routes them to the appropriate backend service. CAS is built on Microsoft Internet Information Services (IIS), a flexible and (presumed) secure web server software designed by Microsoft for Windows to host websites, services, and applications.

The request contained JavaScript to construct an HTTP GET request and this GET request contained a cookie that tricks CAS to route the request to an inappropriate backend host (where `BEResource` is "back end resource")

```
"Cookie": "X-BEResource=localhost~1942062522"
```

Retrospectively, the obvious mistake is that the CAS server is accepting directives to forward requests without sufficient checking and controlling the specification of this forwarding.

Since CVE-2021-26855 was reported fixed, there have been several new variants of SSRF attacks against Exchange. The Exchange API is large and complex, providing a right field for exploit development. Each major release of Exchange has introduced major changes to the interface. This changing attack surface, along with the need to retain a certain amount of backward compatibility, means that the attack surface is also steadily growing.

25.4 Is it an SSRF or Open Reverse Proxy Attack?

The differences between a SSRF attack and an Open Reverse Proxy Attack are subtle and often arbitrary. When a web application is tricked into making a request that it should not have made, then it is usually called an SSRF

[3] `https://nvd.nist.gov/vuln/detail/cve-2021-26855`
[4] `https://attack.mitre.org/groups/G0125`

attack. When a similar attack is directed at some other type of intermediate server, sometimes called a proxy, then the attack is usually called an Open Reverse Proxy Attack. These proxy server might be a web application firewall (WAF), web server (such as NGINX or Apache), or remote access server (such as an SSH server).

No matter which of these terms you use for the attack, the causes are similar, as are the methods of prevention.

25.5 Prevention

Prevention of SSRF attacks can happen at multiple points.

Do not accept requests from unauthenticated connections: It is tempting when deploying a service behind a firewall to be lax on the security settings and assume that any incoming connection is from a host that you control. When your firewalls and other servers are configured correctly, then this assumption holds. However, such assumptions create fragile security and lack defense in depth.

Validate any request that you do forward: Such validate means that both the host to which a request is being sent and the operation being requested must be validated. Validation might be based on an allow list of hosts and operations.

Disable request forwarding when it is not required: If request forwarding is not needed at all, then it should be disabled. If it is never used for particular protocols or network ports, then it should be disabled for those protocols or ports.

In addition, outside of the software domain, security administrators must ensure that firewalls and intrusion detection systems are properly configured.

25.6 Summary

In this chapter, we learned about the causes of server side request forgery attacks, the risks associated with these attacks, the damage that can result from them, and saw two real world examples where SSRF attacks caused serious damage. We also discussed the relationship between an SSRF attack and an open reverse proxy attack. We finished the chapter by discussing ways to prevent these attacks.

25.7 Exercises

1. In this chapter, we mentioned some major websites that have been vulnerable to SSRF attacks. Choose one of these attacks, or another that you have found information about, and research the details. Try to find enough information that you can understand how the attacker conducted the attack and what they were trying to

accomplish. Note that information on such attacks is often incomplete or even contradictory.

2. Is a service that is deployed in the cloud more or less secure than one hosted on your local hardware? Explain your answer.
3. Assume that your organization detected that an SSRF attack was in progress. Or that someone inside your organization discovered that one of your servers was vulnerable to a SSRF attack.
    a. What is the first step that you would take to stop the attack or protect the server?
    b. What steps would you next take?