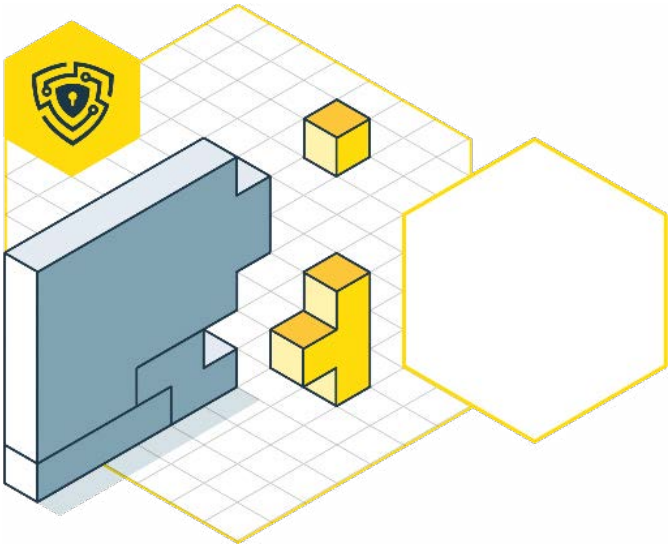
CRASHTEST **SECURITY**

GUIDE FOR SERVER-SIDE REQUEST FORGERY ATTACK PREVENTION

WHAT ARE THE STEPS TO KEEP YOUR WEB APP OR API SAFE FROM SUCH VULNERABILITY

GUIDE TO SSRF VULNERABILITY PREVENTION

Table of Contents



INTRODUCTION TO THIS GUIDE

The Server Application Programming Interface (SAPI) allows web servers to process information from external networks through server-side requests. These server-side requests enable the application server to -- read from or write to -- external systems. While the seamless information exchange offers several benefits for modern application delivery, misconfigurations often lead to a vulnerable web application where malicious actors can modify the target resource URL and inject unexpected user inputs to obtain unauthorized access to sensitive data. Known as server-side request forgery (SSRF), the attack follows a common pattern where hackers trick the target application into making malicious requests to unintended external resources.

This guide discusses the server-side request forgery vulnerability, its impacts, prevention techniques, and best practices to mitigate such attacks.

WHAT IS SERVER-SIDE REQUEST FORGERY?

Server-side request forgery attacks are orchestrated mainly by inducing the server-side application to make malicious requests. This allows a malicious actor to obtain unauthorized access to restricted internal services and connect with arbitrary external entities, potentially exfiltrating sensitive data. By modifying the URL parameter, attackers can also read the server's configuration settings and connect to internal systems that are not intended for exposure. With this attack, hackers can compromise the application itself or other backend systems with which the target server communicates.

Most server-side request forgery attack vectors are easy to find since each application's traffic flow includes URL parameters within the request body. Some standard techniques that attackers use to uncover these vulnerabilities include:

- Partial request URLs - In some vulnerable web servers, the application only includes a partial path in the request's URL parameters. This value is parsed on the server side and incorporated into a full URL. Attackers can recognize this value as a URL path and modify it, enabling the server to make malicious requests.
- Inclusion of URLs in data formats - Some applications rely on data formats that enable the data parser to allow the inclusion of URLs, making them susceptible to attacks. For instance, if an application receives data in XML format and parses it, attackers might include External XML Entities (XXE) in an incoming request, creating an SSRF attack vector.
- Request forgery via the referrer header - In applications that employ analytic solutions to track users, the application server logs the referrer header to track incoming links. In such instances, the analytics solution includes links in the header to visit and analyze the contents of other third-party sites. These referrer headers offer attack surfaces that allow a malicious actor to obtain and alter a legitimate user's incoming request.

SSRF attacks are typically categorized into:

- Blind SSRF attacks - These attacks exploit vulnerabilities that allow attackers to issue a server-side request to a URL. Still, the response to the request is not reflected in the application's client-side response. While these vulnerabilities are harder to exploit, a successful attack often leads to severe consequences, including remote code execution on backend systems.
- Direct SSRF attacks - In this type of attack, the hacker tricks the web application into issuing a server-side request and obtains the contents of this server-side response through the application's client-side response. Attackers can use this response to compromise the vulnerable server itself or other backend systems connected to it.

SSRF attacks can target almost all public-facing servers that access resources from external systems without validating user-supplied URLs. The vulnerable web server submitting the request is automatically assumed to be trusted. This allows the external attacker to request targets outside the internal network even when protected by application layer controls, firewall policies, network access control rules, or a VPN.

SSRF VULNERABILITY - SEVERITY LEVEL

The SSRF vulnerability is ranked number 10 on the OWASP 2021 Top 10 list of vulnerabilities. The vulnerability has a relatively low attack incidence rate of 2.72% since exploiting it requires an application that does not validate user-controlled data while using server-side requests to access resources. On account of the multiple ways to circumvent application layer controls against SSRF, the vulnerability, on the other hand, has a high average weighted exploit of 8.28.

Some common approaches to circumvent application layer controls include:

- Blacklist-based input filters - Some applications employ a block-list approach, where the application blocks request going to specific hostnames, IP addresses, and sensitive URLs. Attackers can circumvent these filters using various methods, such as:

  - Using alternative IP address representations
  - Registering their own domain name that resolves to a blacklisted target address
  - Using case variation or URL obfuscation to obfuscate blacklisted strings

- Whitelist-based input filters - On applications that follow the whitelist approach, where the server only allows outbound traffic to match a list of specified values, attackers can use multiple approaches to circumvent whitelists, such as:

  - Use of special characters to embed information in URLs
  - Leveraging wildcard DNS services to place malicious inputs to qualified DNS names
  - Confusing the URL parser with URL encoded characters

The server-side request forgery vulnerability has a relatively high average weighted impact of 6.72. Some effects of a successful SSRF attack include the following:

- Sensitive data exposure - This approach allows attackers to exploit a target URL and exfiltrate data from services that should not be directly exposed to the internet. Some of such services include metadata storage services, database HTTP interfaces, internal REST interfaces, and files.

- Remote code execution - An attacker can leverage input validation errors to inject malicious code into the server that only expects to read data from trusted sources.

- Cross-site port attacks - Some responses to the server-side request allow an attacker to obtain system-level information of the target server. For instance, data on the server's response time may reveal whether the request was processed successfully. Such instances are common targets of cross-site port attacks where hackers can also use port scans to identify good host-port pairs for orchestrating deeper, system-level attacks.

- Denial of Service attacks - These are orchestrated by manipulating backend systems and flooding the target server with large requests, resulting in a server crash. Most internal servers do not support large amounts of traffic and are susceptible to denial-of-service attacks.

Being a new entrant to the OWASP Top 10, the SSRF vulnerability is only mapped to one Common Weakness Enumeration (CWE-918: Server-Side Request Forgery). With 385 mapped CVEs and 9503 total occurrences, the vulnerability is commonly found in modern web systems and is attributed to an average coverage of 67.72%.

HOW TO IDENTIFY SSRF WITH CRASHTEST SECURITY?

Crashtest Security helps reduce security risks through automated penetration testing and vulnerability scanning. The platform offers a suite of vulnerability scanners that helps detect vulnerabilities and misconfigurations. Vulnerability scanners provided by the Crashtest Security Suite include:

- Microservices scanner - this scanner evaluates inbound and outbound traffic between microservices to ensure no modifications can be done to service requests. The scanner also identifies other vulnerabilities in the microservices that can allow a malicious actor to request other internal services, providing further defense against SSRF attacks.

- HTTP header scanner - Applications use special request headers such as host headers and referrer headers to enable specific functionality for allowing access to resources. Malicious attackers craft header injection attacks to include unintended information within these headers. Crashtest Security's HTTP header scanner helps identify and remediate all host header injection vulnerabilities, including those that can lead to SSRF attacks, such as cross-site port attacks and open redirection exploits.

- XXE vulnerability scanner - The XXE vulnerability scanner helps developers detect issues before attackers leverage External XML Entity (XXE) vulnerabilities to perform SSRF attacks in production.

- URL fuzzer scanner - The scanner helps security analysts find resource files, routes, and directories that are sensitive, hidden, or susceptible to SSRF attacks. This website directory scanner prevents sensitive data exposure and the exfiltration of information that can be used to compromise an entire system.

- OWASP scanner - This scanner performs benchmark tests against all vulnerabilities, including SSRF (A10: 2021), identified by the Online Web Application Security Project.

Crashtest Security also enables security teams to perform ethical hacks and black-box penetration tests to simulate scenarios and assess how attackers leverage SSRF vulnerabilities for attacks. The platform also outputs actionable reports that outline security levels and remediation advice to help security administrators adopt best practices against SSRF attacks.

BEST PRACTICES IN PREVENTING SECURITY LOGGING AND MONITORING FAILURES

Security measures to prevent server-side request forgery (SSRF) attacks include:

STRICT ACCESS CONTROLS

Robust network access control rules prevent attackers from exploiting an organization's internal networks and submitting malicious requests. Enforcing access controls with multi-factor authentication, role-based authorizations, or other rule-based security measures at the network perimeter restricts attackers from identifying and exploiting SSRF vulnerabilities.

When implementing access controls, organizations should also consider the following and administer rules accordingly for robust security:

- who should have access to what data and systems?
- what level of access does each user need?
- how will users be authenticated?
- how will authorization be granted?
- how will access be monitored and audited?

FIREWALL POLICIES

As the first line of defense, organizations can prevent SSRF attacks by implementing firewall policies defining the external servers that an application can connect to. The policies can either be applied at specific points at the network level or closer to the host using access control rules at the machine's loopback network interface.

WHITELISTS AND DNS RESOLUTION

A common approach to combating SSRF attacks is to whitelist all the DNS names and decimal IP addresses the server should connect to. The whitelist should also apply to user-controllable inputs, ensuring the application only accepts known requests. A whitelist approach enforces stricter control over server-side requests, as the application can only accept, bind and transmit content following a pre-configured standard.

RESPONSE HANDLING

The backend server should be restricted from sending raw response bodies to clients after receiving a response from an external server. The server should verify the service response to ensure it only includes valid data types and does not expose any sensitive information before delivering it to the client side.

BEST PRACTICES IN PREVENTING SSRF VULNERABILITIES

Some recommended practices to prevent SSRF vulnerabilities include:

DISABLE UNUSED URL SCHEMAS

The application server should strictly accept the input schema that is currently being used for making requests while discarding the rest. This helps prevent SSRF vulnerabilities since it makes it difficult for attackers to craft malicious requests and submit them with their own URLs. Attackers commonly exploit the file://, ftp://, gopher://, and dict:// URL schemas for SSRF attacks as they enable directory and administrative ports access, allowing them to craft malicious server-side requests. It is recommended instead to use the https:// schema that enforces transport layer security and prevents attackers from accessing internal resources even with access to the network.

ENFORCE INPUT SANITIZATION AND VALIDATION

A common approach to exploiting SSRF vulnerabilities is manipulating the application through user-controllable input and making a malicious request. The application should never trust any incoming input by default to avoid this. Additionally, all incoming inputs should be sanitized to remove unexpected characters to follow a standardized format and ensure no malicious code or commands are injected into the system.

PERFORM AUTHENTICATION ON ALL INTERNAL

Some services, such as MongoDB, ElasticSearch, and MemCached, do not require additional authentication to process requests. In such instances, an attacker can exploit a vulnerable server to craft malicious requests and obtain unauthorized access to such services. To keep configuration information and sensitive data secure, it is important to secure these services through an additional layer of user authentication and authorization.

PREVENT SSRF ATTACKS WITH CRASHTEST SECURITY

Crashtest Security helps administer an automatic scanning and testing framework to prevent server-side request forgery vulnerabilities and other modern web security risks. The platform continuously benchmarks applications against the OWASP Top 10 vulnerabilities to help mitigate critical security risks through proactive detection, identification, and remediation.

Crashtest Security seamlessly integrates security testing into development workflows to ensure threats are detected and remediated since the early stages of the SDLC. With its quick security assessment and actionable security reports, cross-functional teams can identify security blind spots and remediate threats faster.

To know more about how Crashtest Security can help eliminate SSRF vulnerabilities before they are exploited in production, try a free, 14-day demo here.

Start 2-Week Trial for Free

CRASHTEST **SECURITY**

WWW.CRASHTEST-SECURITY.COM