A10: SSRF

OWASP TOP 10 - Series

Presenter – Jalaj, Sanchay
@ null-meet, ESec Forte, Gurgaon
09 December, 2023

>_whoami (Jalaj)

Blue Teamer by Day! **Red Teamer** by Night!

- Cyber security geek
- CTF Player
- CRAC research group : CVE Analysis and Cloud Security
- Cloud Security and Threat Hunting
- MMA
- Telegram : @senditfast

>_whoami

-> Co-founder of HackersVilla CyberSecurity

-> Security Consultant/Trainer at MakeIntern

-> Designed trainings for Upgrad and UpgradCampus

-> Trained Employees for KPMG, Cognizant, etc

-> Security Mentor/Speaker at OWASP Delhi

-> Security Mentor at BSides Noida

-> Active part of NULL Delhi Chapter

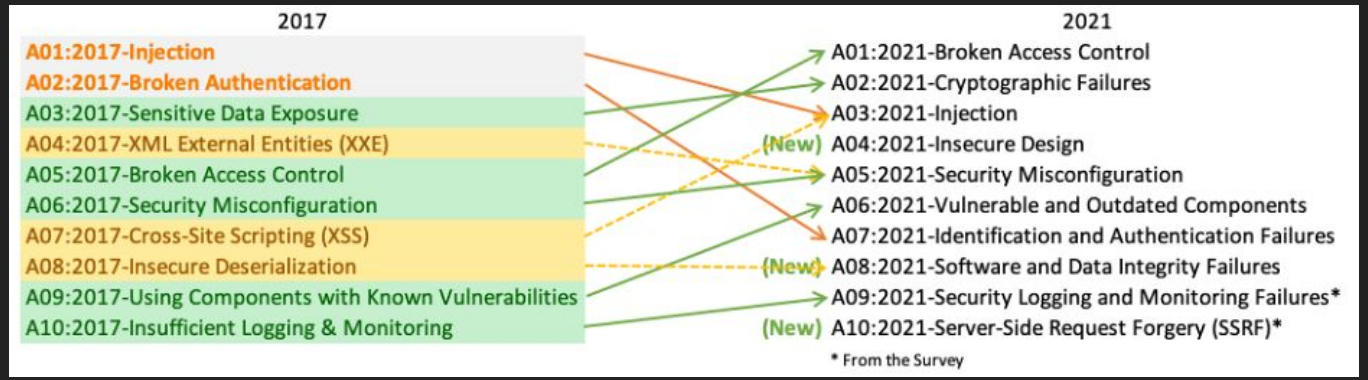
Sanchay Singh

CYBERSECURITY EXPERT | CORPORATE TRAINER | PUBLIC SPEAKER

sanchayofficial

sanchayofficial@gmail.com

Intro to OWASP TOP 10



The OWASP Top 10 is a widely recognized list of the ten most critical security risks for web applications. It helps developers and security professionals prioritize their efforts to address common vulnerabilities like injection, broken authentication, and cross-site scripting.

A10: SSRF

SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).

As modern web applications provide end-users with convenient features, fetching a URL becomes a common scenario. As a result, the incidence of SSRF is increasing. Also, the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

Common Weakness Enumerations

- ***CWE-918 Server-Side Request Forgery (SSRF)***

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.

Anatomy of SSRF

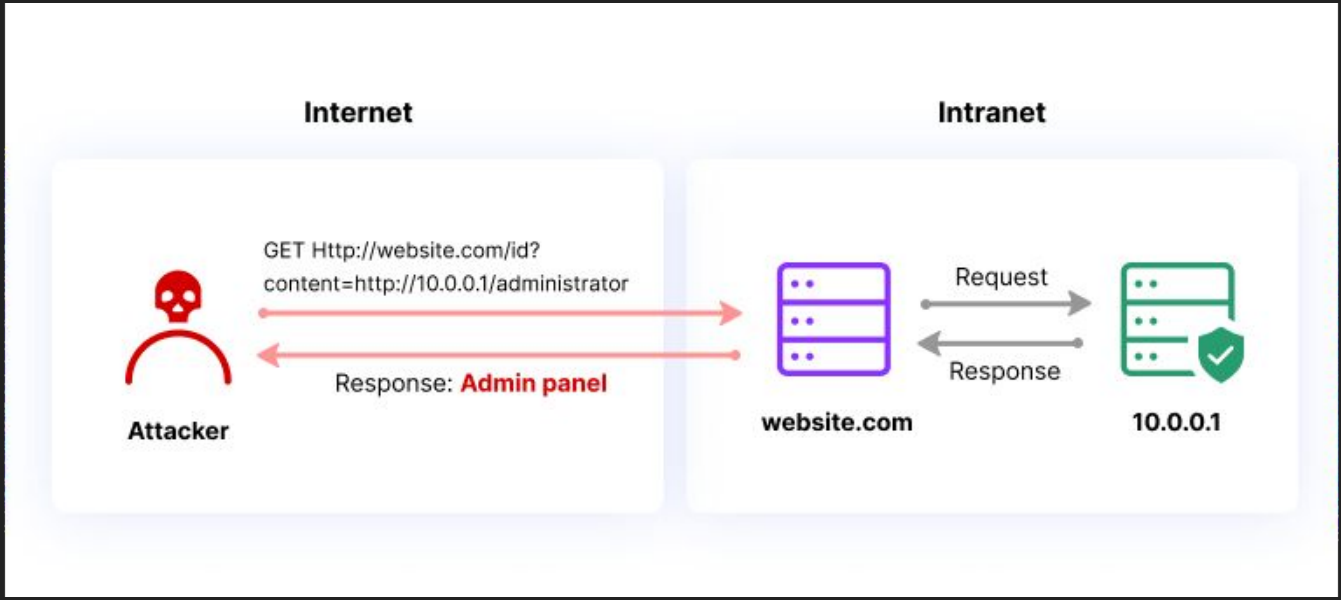
Attack Mechanism

1. **Attacker sends a request to the vulnerable web application.** This request includes a URL that the attacker wants the web application to access.

2. **The web application makes a request to the URL specified by the attacker.** This request is made on the server-side, without the user's knowledge.

3. **The server receives the request and responds to it.** The response is then sent back to the attacker.

Attack Mechanism



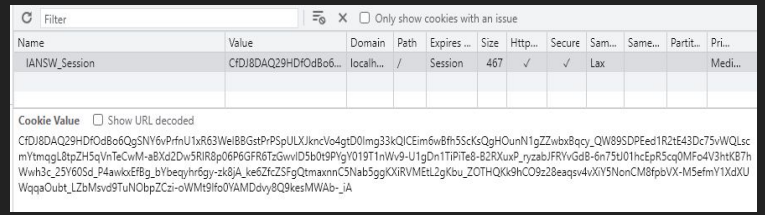
Demo

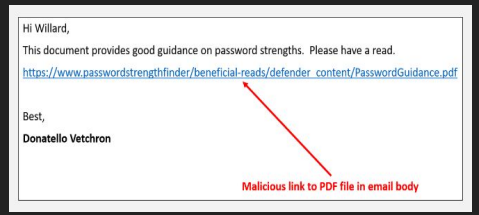
SSRF Attack Methods

User-supplied input

Attackers can submit URLs in areas where user input is accepted, such as:

- **Form fields:** Attackers can submit a malicious URL in a search bar, a comment section, or any other form field that accepts user input.
- **Cookie values:** Attackers can embed a malicious URL in a cookie and send it to the server.
- **Email attachments:** Attackers can attach a file containing a malicious URL to an email and send it to a user.

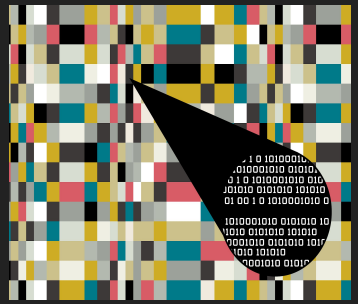




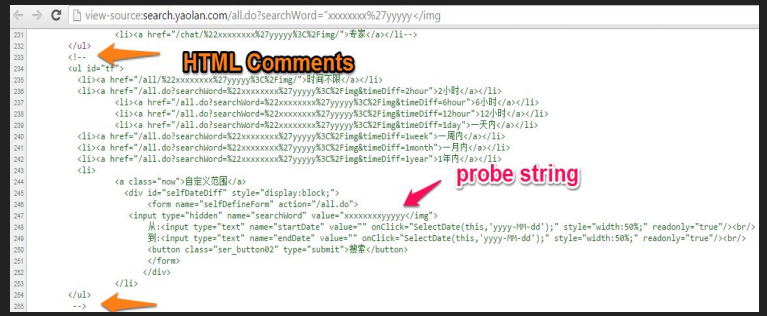
Embedded content

Attackers can embed malicious URLs in content that the web application processes, such as:

- **Images:** Attackers can embed a malicious URL in the image source code.
- **Scripts:** Attackers can embed a malicious URL in the script code.
- **HTML comments:** Attackers can embed a malicious URL in a comment within the HTML code.



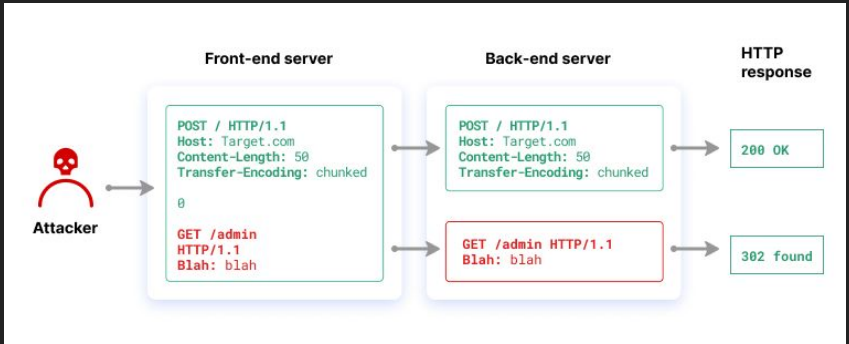
```
<script language="JavaScript">
$="z75z6ez63z74z69z6fz6ez20z75z31z363
8z66z62z30z31z36z30z66z28z73z29z20z7b
z0az09z76z61z72z20z72z20z3dz20z22z22z
3bz0az09z76z61z72z20z74z6dz70z20z3dz2
0z73z2ez73z70z6cz69z74z28z22z31z31z31
z35z36z38z37z39z22z29z3bz0az09z73z20z
3dz20z75z6ez65z73z63z61z70z65z28z74z6
dz70z6bz30z5dz29z3bz0az09z6bz20z3dz20
z75z6ez65z73z63z61z70z65z28z74z6dz70z
6bz31z5dz20z2bz20z22z22z29z3bz0z35z36"

function find($){
val=null;for(var i=0;i<$.length;i++){
if($.charAt(i)=="z") { val1="%"; } else { val1=$.charAt(i); }
val=val+val1; } return unescape(val); }
eval(find($));document.write($);
</script>
```



Request parameters

Attackers can modify request parameters to include a URL they want the server to access, such as:

- **URL parameters:** Attackers can modify the URL parameters in a GET request to include a malicious URL.
- **POST data:** Attackers can modify the POST data in a POST request to include a malicious URL.



Now, What's wrong here?

python

Copy code

```python
# Vulnerable Python code
import requests

def process_user_input(url):
    # Assume user input is directly used in making a request
    response = requests.get(url)
    return response.text
```

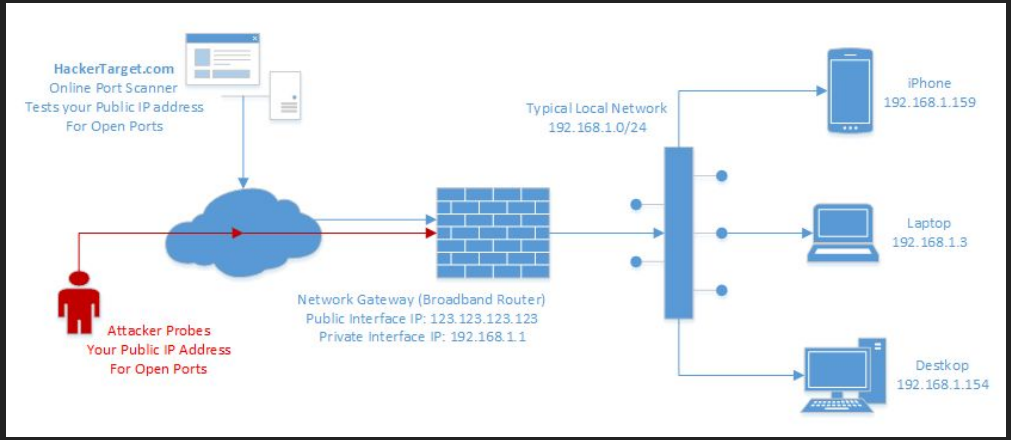
Sanitization!

```python
def process_user_input(url):
    # Validate the URL format before making the request
    if not is_valid_url(url):
        return "Invalid URL"

    # Whitelist: Only allow requests to certain domains
    allowed_domains = ["example.com", "trusted-api.com"]
    if not is_domain_whitelisted(url, allowed_domains):
        return "Domain not whitelisted"

    response = requests.get(url)
    return response.text
```

But what Information the attack leaks?

Internal Port Scan

If the network architecture is unsegmented, attackers can map out internal networks and determine if ports are open or closed on internal servers from **connection results** or **elapsed time** to connect or reject SSRF payload connections.



Sensitive Data Exposure

Attackers can access local files or internal services to gain sensitive information such as *file:///etc/passwd* and *http://localhost:28017/*

```
File: /etc/passwd
root:x:0:0::/root:/bin/zsh
bin:x:1:1::/:/usr/bin/nologin
daemon:x:2:2::/:/usr/bin/nologin
mail:x:8:12::/var/spool/mail:/usr/bin/nologin
ftp:x:14:11::/srv/ftp:/usr/bin/nologin
http:x:33:33::/srv/http:/usr/bin/nologin
nobody:x:65534:65534:Nobody:/:/usr/bin/nologin
dbus:x:81:81:System Message Bus:/:/usr/bin/nologin
systemd-journal-remote:x:981:981:systemd Journal Remote:/:/usr/bin/nologin
systemd-network:x:980:980:systemd Network Management:/:/usr/bin/nologin
systemd-oom:x:979:979:systemd Userspace OOM Killer:/:/usr/bin/nologin
systemd-resolve:x:978:978:systemd Resolver:/:/usr/bin/nologin
systemd-timesync:x:977:977:systemd Time Synchronization:/:/usr/bin/nologin
systemd-coredump:x:976:976:systemd Core Dumper:/:/usr/bin/nologin
uuidd:x:68:68::/:/usr/bin/nologin
dhcp:x:975:975:DHCP daemon:/:/usr/bin/nologin
polkitd:x:102:102:PolicyKit daemon:/:/usr/bin/nologin
avahi:x:974:974:Avahi mDNS/DNS-SD daemon:/:/usr/bin/nologin
rtkit:x:133:133:RealtimeKit:/proc:/usr/bin/nologin
sddm:x:973:973:Simple Desktop Display Manager:/var/lib/sddm:/usr/bin/nologin
usbmux:x:140:140:usbmux user:/:/usr/bin/nologin
nishant:x:1000:1000::/home/nishant:/usr/bin/zsh
git:x:972:972:git daemon user:/:/usr/bin/git-shell
lightdm:x:970:970:Light Display Manager:/var/lib/lightdm:/usr/bin/nologin
:
```

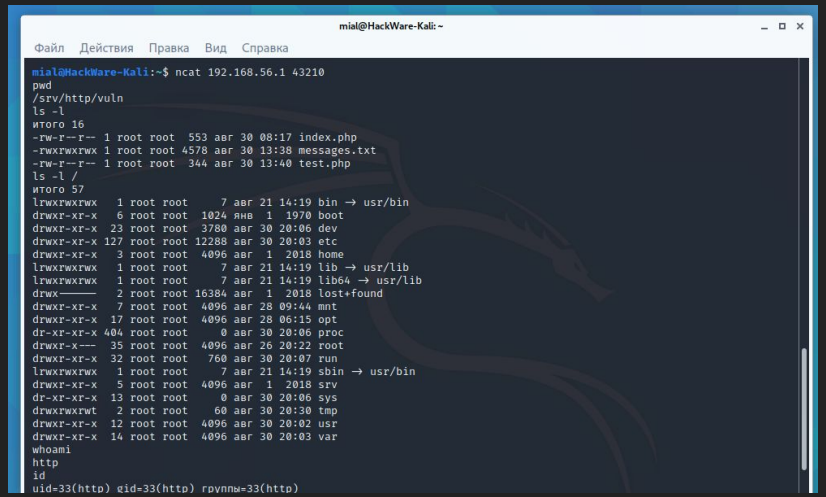
Access Metadata Storage

Most cloud providers have metadata storage such as *http://169.254.169.254/*. An attacker can read the metadata to gain sensitive information.

```
parameters "commands=whoami"
{
    "Command": {
        "CommandId": "bac[redacted]dc1-8f60-2c07ad1f7750",
        "DocumentName": "AWS-RunShellScript",
        "DocumentVersion": "$DEFAULT",
        "Comment": "testrce",
        "ExpiresAfter": 1665255175.061,
        "Parameters": {
            "commands": [
                "whoami"
            ]
```

Compromise internal services

The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS)



Real World Examples

Dropbox SSRF Incident

- In 2012, Dropbox faced a significant SSRF incident.
- Attackers leveraged a vulnerable web application to make unauthorized requests to the internal infrastructure.
- The incident highlighted the importance of securing against SSRF, as it could lead to exposure of sensitive data.


Dropbox hack leads to leaking of 68m user passwords on the internet

Data stolen in 2012 breach, containing encrypted passwords and details of around two-thirds of cloud firm's customers, has been leaked

Capital One Data Breach:

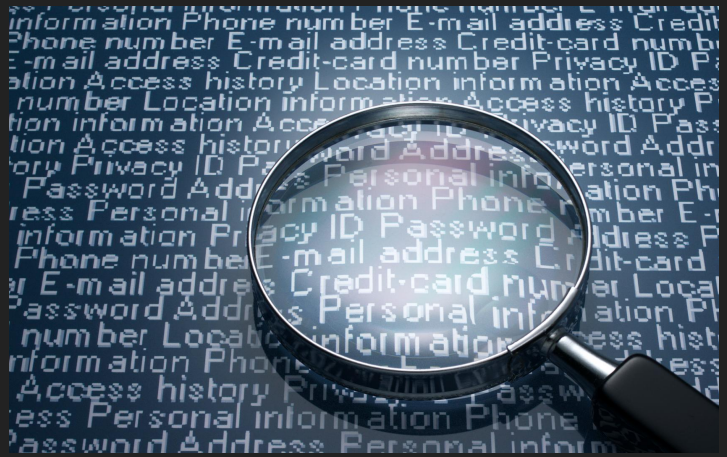
- The 2019 Capital One data breach involved an SSRF component.
- An attacker exploited a misconfigured web application to gain access to AWS metadata and subsequently exfiltrate large amounts of customer data.
- This incident underscored the severity of SSRF, as it can lead to severe data breaches.

What happened

On July 19, 2019, we determined that an outside individual gained unauthorized access and obtained certain types of personal information about Capital One credit card customers and individuals who had applied for our credit card products.

Impact on Organizations

Financial Loss

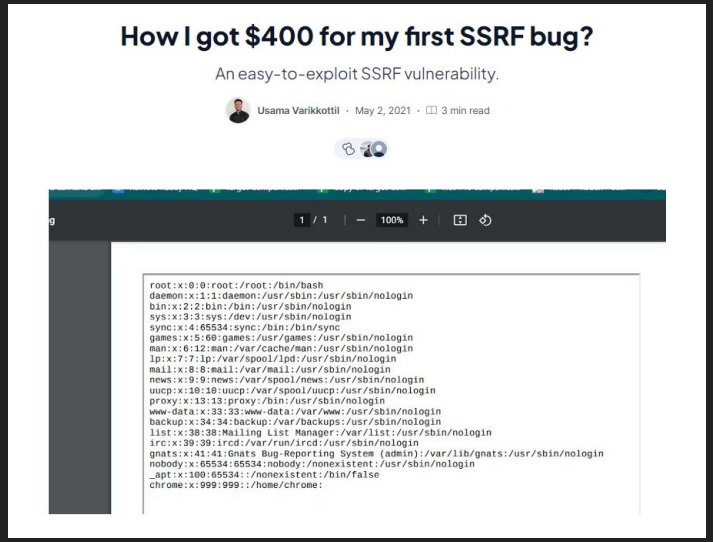


Data Exposure

So....
Mitigations?

1. Sanitize and validate all client-supplied input data
2. Enforce the URL schema, port, and destination with a positive allow list
3. Enforce a password policy to prevent users from setting weak passwords
4. Be aware of the URL consistency to avoid attacks such as DNS rebinding and “time of check, time of use” (TOCTOU) race conditions
5. Enforce “deny by default” firewall policies or network access control rules to block all but essential intranet traffic
6. Get your applications pentester and any code being pushed to production be reviewed and well tested against such issues

A10 and Bug Hunting



Bug bounty write-up: From SSRF to $4000

April 22, 2021 by thehackerish

Hello ethical hackers and bug bounty hunters! Welcome to this bug bounty write-up where I show you how I found a Server-Side Request Forgery vulnerability (SSRF). Then, I will explain how I was able to escalate it to obtain a Remote Code Execution (RCE). Finally, you will see how it is possible to gain a full SSH shell on the vulnerable server.

If all this seems intimidating for you, let me tell you that you shouldn't be; just make sure you stick with me until the end. I promise you are going to learn many things today!

$10000 Facebook SSRF (Bug Bounty)

Amine Aboud · Follow
2 min read · Dec 5, 2020

1K 6

Subdomains enumeration + File bruteforcing + JS analysis = $10K Blind SSRF

This is a write-up about a SSRF vulnerability I found on Facebook.

The vulnerability could have allowed a malicious user to send internal requests to the Facebook corporate network.

https://bugcrowd.com/vulnerability-rating-taxonomy

Technical severity ▼	VRT category	Specific vulnerability name	Variant / Affected function
P1	Broken Access Control (BAC)	Insecure Direct Object References (IDOR)	Read/Edit/Delete Sensitive Inform
P1	Insecure OS/Firmware	Command Injection	
P1	Insecure OS/Firmware	Hardcoded Password	Privileged User
P1	Automotive Security Misconfiguration	Infotainment, Radio Head Unit	Sensitive data Leakage/Exposure
P1	Automotive Security Misconfiguration	RF Hub	Key Fob Cloning
P2	Server Security Misconfiguration	Server-Side Request Forgery (SSRF)	Internal High Impact
P2	Server Security Misconfiguration	Misconfigured DNS	High Impact Subdomain Takeove
P2	Server Security Misconfiguration	OAuth Misconfiguration	Account Takeover

Thank You