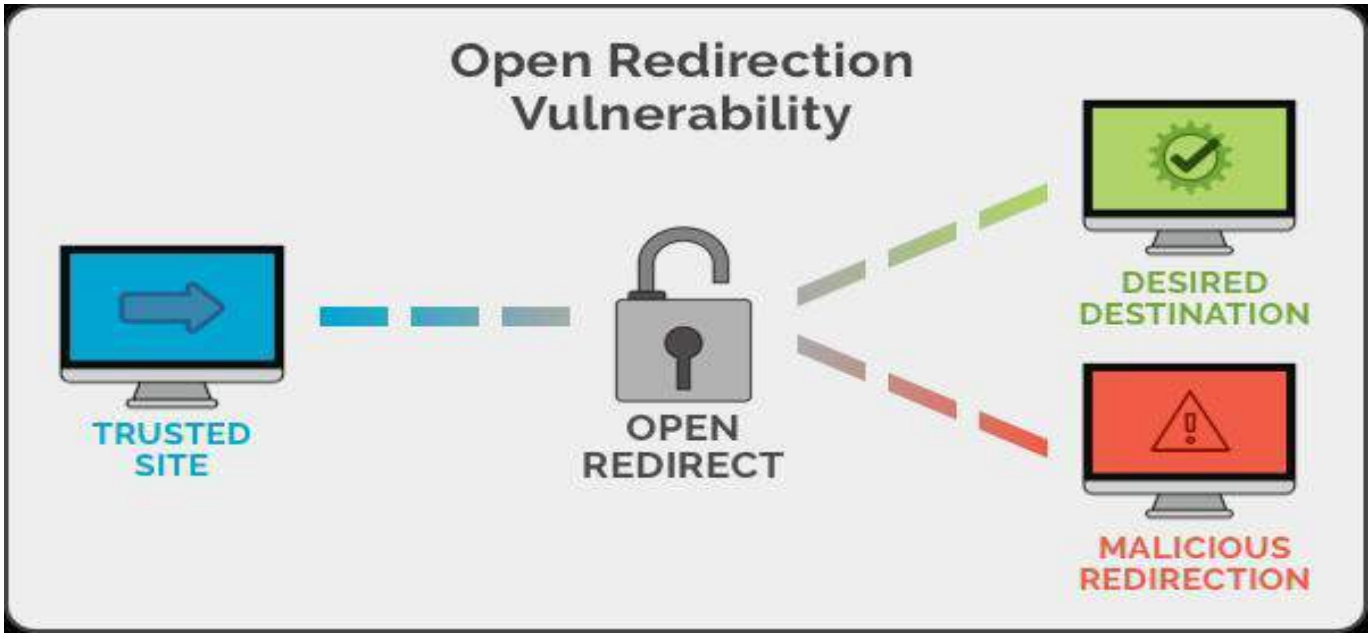
OPEN REDIRECTION

URL Redirection is a vulnerability which allows an attacker to force users of your application to an untrusted external site.



Parameter based URL redirection is the most common and easy to spot. There are two behaviors which contribute to this issue:

- A **GET** parameter containing a URL/URI.
- A **302/301 redirect** made using that parameter.

So if you see a parameter passed in a URL before a page redirection, it's a good idea to test if that can be modified with an arbitrary URL.

Most sites usually redirect the user after some type of action such as **logging in, logging out, password change, signup.**

The parameter can usually be found in the URL, or sometimes you need to hunt in .js files for referenced parameters. It is highly likely that the login page will handle some type of redirect parameter.

Example -

The following code obtains a URL from the query string and then redirects the user to that URL.

```
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
```

The problem with the above code is that an attacker could use this page as part of a phishing scam by redirecting users to a malicious site. For example, assume the above code is in the file example.php. An attacker could supply a user with the following link:

(Attack code)

http://example.com/example.php?url=http://malicious.example.com

The user sees the link pointing to the original trusted site (example.com) and does not realize the redirection that could take place.

Safe URL Redirects

When we want to redirect a user automatically to another page (without an action of the visitor such as clicking on a hyperlink) you might implement a code such as the following:

Java

```
Response.sendRedirect(http://www.mysite.com);
```

PHP

```
<?php
/* Redirect browser */
Header("Location: http://www.mysite.com");
/* Exit to prevent the rest of the code from executing */
Exit;
?>
```

In the examples above, the URL is being explicitly declared in the code and cannot be manipulated by an attacker.

Dangerous URL Redirects

The following examples demonstrate unsafe redirect and forward code.

The following **Java** code receives the URL from the parameter named url (GET or POST) and redirects to that URL:

- Response.sendRedirect(request.getParameter("url"));

The following **PHP** code obtains a URL from the query string (via the parameter named url) and then redirects the user to that URL. Additionally, the PHP code after this header() function will continue to execute, so if the user configures their browser to ignore the redirect, they may be able to access the rest of the page.

- $redirect_url = $_GET['url'];
- Header("Location: " . $redirect_url);

The above code is vulnerable to an attack if no validation or extra method controls are applied to verify the certainty of the URL.

Common injection points / parameters

- /{payload}
- ?next={payload}
- ?url={payload}
- ?target={payload}
- ?rurl={payload}
- ?dest={payload}
- ?destination={payload}
- ?redir={payload}
- ?redirect_uri={payload}
- ?redirect_url={payload}
- ?redirect={payload}
- /redirect/{payload}
- /cgi-bin/redirect.cgi?{payload}
- /out/{payload}
- /out?{payload}
- ?view={payload}
- /login?to={payload}
- ?image_url={payload}
- ?go={payload}
- ?return={payload}
- ?returnTo={payload}
- ?return_to={payload}
- ?checkout_url={payload}
- ?continue={payload}
- ?return_path={payload}
- “returnUrl”
- “returnUri”
- “locationUrl”
- “goTo”
- “ref=”
- “referrer=”
- “backUrl”
- “successUrl”

What Is the Impact of an Open Redirection Vulnerability?

As mentioned above, the impacts can be many, and vary from theft of information and credentials, to the redirection to malicious websites containing attacker-controlled content, which in some cases even cause XSS attacks. So even though an open redirection might sound harmless at first, the impacts of it can be severe should it be exploitable.

How can you prevent open redirection vulnerabilities?

1. The easiest and most effective way to prevent vulnerable open redirects would be to-
2. Not let the user control where your page redirects them to. If you have to redirect the user based on URLs, instead of using untrusted input you should **always use an ID** which is **internally resolved** to the respective URL.
3. If you want the user to be able to issue redirects you should use a redirection page that requires the **user to click on the link instead of just redirecting them**.
4. You should also check that the URL begins with http:// or https:// and also **invalidate all other URLs** to prevent the use of malicious schemes such as javascript:.