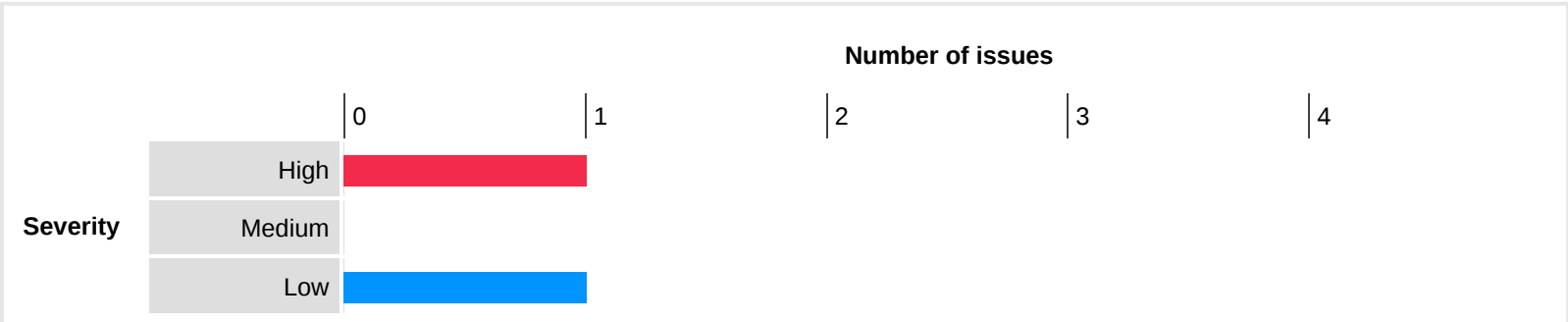


Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low, Information or False Positive. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	1	0	0	1
	Medium	0	0	0	0
	Low	1	0	0	1
	Information	0	2	1	3
	False Positive	0	0	0	0

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.




Contents

- 1. Cleartext submission of password
- 2. Unencrypted communications
- 3. Path-relative style sheet import
- 4. Cross-site request forgery
- 5. Frameable response (potential Clickjacking)

1. Cleartext submission of password

Summary

	Severity:	High
	Confidence:	Certain
	Host:	http://localhost
	Path:	/DVWA/login.php

Issue detail

The page contains a form with the following action URL, which is submitted over clear-text HTTP:

- http://localhost/DVWA/login.php

The form contains the following password field:

- password

Issue background

Some applications transmit passwords over unencrypted connections, making them vulnerable to interception. To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Vulnerabilities that result in the disclosure of users' passwords can result in compromises that are extremely difficult to investigate due to obscured audit trails. Even if the application itself only handles non-sensitive information, exposing passwords puts users who have re-used their password elsewhere at risk.

Issue remediation

Applications should use transport-level encryption (SSL or TLS) to protect all sensitive communications passing between the client and the server. Communications that should be protected include the login mechanism and related functionality, and any functions where sensitive data can be accessed or privileged actions can be performed. These areas should employ their own session handling mechanism, and the session tokens used should never be transmitted over unencrypted communications. If HTTP cookies are used for transmitting session tokens, then the secure flag should be set to prevent transmission over clear-text HTTP.

Vulnerability classifications

- [CWE-319: Cleartext Transmission of Sensitive Information](#)
- [CAPEC-117: Interception](#)

Request

GET /DVWA/login.php HTTP/1.1 Host: localhost Accept-Encoding: gzip, deflate Accept: */* Accept-Language: en-US;q=0.9,en;q=0.8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36 Connection: close Cache-Control: max-age=0

Response

HTTP/1.1 200 OK
Date: Fri, 19 Apr 2024 04:05:15 GMT
Server: Apache/2.4.46 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=uuvsjct7tr7o1pfa22o5trvsdr; expires=Sat, 20-Apr-2024 04:05:15 GMT; Max-Age=86400; path=/; domain=localhost; HttpOnly; SameSite=1
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1415
Connection: close
Content-Type: text/html; charset=utf-8

```
<!DOCTYPE html>

<html lang="en-GB">

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />


    <title>Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Developm
...[SNIP]...
<div id="content">

  <form action="login.php" method="post">

    <fieldset>
...[SNIP]...
</label> <input type="password" class="loginInput" AUTOCOMPLETE="off" size="20" name="password"><br />
...[SNIP]...
```

2. Unencrypted communications

Summary

	Severity:	Low
	Confidence:	Certain
	Host:	http://localhost
	Path:	/

Issue description

The application allows users to connect to it over unencrypted connections. An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. Furthermore, an attacker able to modify traffic could use the application as a platform for attacks against its users and third-party websites. Unencrypted connections have been exploited by ISPs and governments to track users, and to inject adverts and malicious JavaScript. Due to these concerns, web browser vendors are planning to visually flag unencrypted connections as hazardous.

To exploit this vulnerability, an attacker must be suitably positioned to eavesdrop on the victim's network traffic. This scenario typically occurs when a client communicates with the server over an insecure connection such as public Wi-Fi, or a corporate or home network that is shared with a compromised computer. Common defenses such as switched networks are not sufficient to prevent this. An attacker situated in the user's ISP or the application's hosting infrastructure could also perform this attack. Note that an advanced adversary could potentially target any connection made over the Internet's core infrastructure.

Please note that using a mixture of encrypted and unencrypted communications is an ineffective defense against active attackers, because they can easily remove references to encrypted resources when these references are transmitted over an unencrypted connection.

Issue remediation

Applications should use transport-level encryption (SSL/TLS) to protect all communications passing between the client and the server. The Strict-Transport-Security HTTP header should be used to ensure that clients refuse to access the server over an insecure connection.

References

- [Marking HTTP as non-secure](#)
- [Configuring Server-Side SSL/TLS](#)
- [HTTP Strict Transport Security](#)

Vulnerability classifications

- [CWE-326: Inadequate Encryption Strength](#)
- [CAPEC-94: Man in the Middle Attack](#)
- [CAPEC-157: Sniffing Attacks](#)

3. Path-relative style sheet import

Summary

	Severity:	Information
	Confidence:	Firm
	Host:	http://localhost
	Path:	/DVWA/login.php

Issue detail

The application may be vulnerable to path-relative style sheet import (PRSSI) attacks. The first four conditions for an exploitable vulnerability are present (see issue background):

1. The original response contains a path-relative style sheet import (see response 1).
2. When superfluous path-like data is placed into the URL following the original filename (see request 2), the application's response still contains a path-relative style sheet import (see response 2).
3. Response 2 can be made to render in a browser's quirks mode. Although the page contains a modern doctype directive, the response does not prevent itself from being framed. An attacker can frame the response within a page that they control, to force it to be rendered in quirks mode. (Note that this technique is IE-specific and due to P3P restrictions might sometimes limit the impact of a successful attack.)
4. When the path-relative style sheet import in response 2 is requested (see request 3) the application returns something other than the CSS response that was supposed to be imported (see response 3).

It was not verified whether condition 5 holds (see issue background), and you should manually investigate whether it is possible to manipulate some text within response 3, to enable full exploitation of this issue.

Issue background

Path-relative style sheet import vulnerabilities arise when the following conditions hold:

1. A response contains a style sheet import that uses a path-relative URL (for example, the page at "/original-path/file.php" might import "styles/main.css").
2. When handling requests, the application or platform tolerates superfluous path-like data following the original filename in the URL (for example, "/original-path/file.php/extra-junk/"). When superfluous data is added to the original URL, the application's response still contains a path-relative stylesheet import.
3. The response in condition 2 can be made to render in a browser's quirks mode, either because it has a missing or old doctype directive, or because it allows itself to be framed by a page under an attacker's control.
4. When a browser requests the style sheet that is imported in the response from the modified URL (using the URL "/original-path/file.php/extra-junk/styles/main.css"), the application returns something other than the CSS response that was supposed to be imported. Given the behavior described in condition 2, this will typically be the same response that was originally returned in condition 1.
5. An attacker has a means of manipulating some text within the response in condition 4, for example because the application stores and displays some past input, or echoes some text within the current URL.

Given the above conditions, an attacker can execute CSS injection within the browser of the target user. The attacker can construct a URL that causes the victim's browser to import as CSS a different URL than normal, containing text that the attacker can manipulate.

Being able to inject arbitrary CSS into the victim's browser may enable various attacks, including:

- Executing arbitrary JavaScript using IE's expression() function.
- Using CSS selectors to read parts of the HTML source, which may include sensitive data such as anti-CSRF tokens.
- Capturing any sensitive data within the URL query string by making a further style sheet import to a URL on the attacker's domain, and monitoring the incoming Referer header.

Issue remediation

The root cause of the vulnerability can be resolved by not using path-relative URLs in style sheet imports. Aside from this, attacks can also be prevented by implementing all of the following defensive measures:

- Setting the HTTP response header "X-Frame-Options: deny" in all responses. One method that an attacker can use to make a page render in quirks mode is to frame it within their own page that is rendered in quirks mode. Setting this header prevents the page from being framed.
- Setting a modern doctype (e.g. "<!doctype html>") in all HTML responses. This prevents the page from being rendered in quirks mode (unless it is being framed, as described above).
- Setting the HTTP response header "X-Content-Type-Options: nosniff" in all responses. This prevents the browser from processing a non-CSS response as CSS, even if another page loads the response via a style sheet import.

References

- [Detecting and exploiting path-relative stylesheet import \(PRSSI\) vulnerabilities](#)

Vulnerability classifications

- [CWE-16: Configuration](#)
- [CAPEC-154: Resource Location Spoofing](#)
- [CAPEC-468: Generic Cross-Browser Cross-Domain Theft](#)

Request 1

```
GET /DVWA/login.php HTTP/1.1
Host: localhost
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response 1

```
HTTP/1.1 200 OK
Date: Fri, 19 Apr 2024 04:05:19 GMT
Server: Apache/2.4.46 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=t4211937htu8td634hk4s1s7in; expires=Sat, 20-Apr-2024 04:05:19 GMT; Max-Age=86400; path=/; domain=localhost; HttpOnly; SameSite=1
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1415
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

  <head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<title>Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Developm
...[SNIP]...
</title>

<link rel="stylesheet" type="text/css" href="dvwa/css/login.css" />

</head>
...[SNIP]...
```

Request 2

```
GET /DVWA/login.php/dg7baz/ HTTP/1.1
Host: localhost
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response 2

```
HTTP/1.1 200 OK
Date: Fri, 19 Apr 2024 04:05:34 GMT
Server: Apache/2.4.46 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=llkjmoficl4sq8k9117opembj; expires=Sat, 20-Apr-2024 04:05:34 GMT; Max-Age=86400; path=/; domain=localhost; HttpOnly; SameSite=1
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1415
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <title>Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Developm
...[SNIP]...
</title>

    <link rel="stylesheet" type="text/css" href="dvwa/css/login.css" />

  </head>
...[SNIP]...
```

Request 3

```
GET /DVWA/login.php/dg7baz/dvwa/css/login.css HTTP/1.1
Host: localhost
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Connection: close
```

Response 3

```
HTTP/1.1 200 OK
Date: Fri, 19 Apr 2024 04:05:34 GMT
Server: Apache/2.4.46 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=chfdr4dhgscocvrqa7vrfvsf4h; expires=Sat, 20-Apr-2024 04:05:34 GMT; Max-Age=86400; path=/; domain=localhost; HttpOnly; SameSite=1
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1415
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">


  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <title>Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Developm
...[SNIP]...
```

4. Cross-site request forgery

Summary

	Severity:	Information
	Confidence:	Tentative
	Host:	http://localhost
	Path:	/DVWA/login.php

Issue detail

The request appears to be vulnerable to cross-site request forgery (CSRF) attacks against unauthenticated functionality. This is unlikely to constitute a security vulnerability in its own right, however it may facilitate exploitation of other vulnerabilities affecting application users.

The original request contains parameters that look like they may be anti-CSRF tokens. However the request is successful if these parameters are removed.

Issue background

Cross-site request forgery (CSRF) vulnerabilities may arise when applications rely solely on HTTP cookies to identify the user that has issued a particular request. Because browsers automatically add cookies to requests regardless of their origin, it may be possible for an attacker to create a malicious web site that forges a cross-domain request to the vulnerable application. For a request to be vulnerable to CSRF, the following conditions must hold:

- The request can be issued cross-domain, for example using an HTML form. If the request contains non-standard headers or body content, then it may only be issuable from a page that originated on the same domain.
- The application relies solely on HTTP cookies or Basic Authentication to identify the user that issued the request. If the application places session-related tokens elsewhere within the request, then it may not be vulnerable.
- The request performs some privileged action within the application, which modifies the application's state based on the identity of the issuing user.

- The attacker can determine all the parameters required to construct a request that performs the action. If the request contains any values that the attacker cannot determine or predict, then it is not vulnerable.

Issue remediation

The most effective way to protect against CSRF vulnerabilities is to include within relevant requests an additional token that is not transmitted in a cookie: for example, a parameter in a hidden form field. This additional token should contain sufficient entropy, and be generated using a cryptographic random number generator, such that it is not feasible for an attacker to determine or predict the value of any token that was issued to another user. The token should be associated with the user's session, and the application should validate that the correct token is received before performing any action resulting from the request.

An alternative approach, which may be easier to implement, is to validate that Host and Referer headers in relevant requests are both present and contain the same domain name. However, this approach is somewhat less robust: historically, quirks in browsers and plugins have often enabled attackers to forge cross-domain requests that manipulate these headers to bypass such defenses.

References

- [Web Security Academy: Cross-site request forgery](#)
- [Using Burp to Test for Cross-Site Request Forgery](#)
- [The Deputies Are Still Confused](#)

Vulnerability classifications

- [CWE-352: Cross-Site Request Forgery \(CSRF\)](#)
- [CAPEC-62: Cross Site Request Forgery](#)

Request 1

```
POST /DVWA/login.php HTTP/1.1
Host: localhost
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Connection: close
Cache-Control: max-age=0
Referer: http://localhost/DVWA/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 97
Cookie: security=impossible; PHPSESSID=m8julbdin856lj2p1uie041j0v

username=dOrskfA&password=s2J%21u9t%21F3&Login=Login&user_token=ff0f9a6e42c33df02909a1adae726c14
```

Response 1

```
HTTP/1.1 302 Found
Date: Fri, 19 Apr 2024 04:05:19 GMT
Server: Apache/2.4.46 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: login.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

Request 2

```
POST /DVWA/login.php HTTP/1.1
Host: localhost
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
```


Connection: close
Cache-Control: max-age=0
Referer: http://bbJHp.com/DVWA/login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 53
Cookie: security=impossible; PHPSESSID=m8julbdin856lj2p1uie041j0v

username=dOrskfA&password=s2J%21u9t%21F3&Login=Login

Response 2

HTTP/1.1 302 Found
Date: Fri, 19 Apr 2024 04:05:47 GMT
Server: Apache/2.4.46 (Debian)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: login.php
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8

5. Frameable response (potential Clickjacking)

Summary

	Severity:	Information
	Confidence:	Firm
	Host:	http://localhost
	Path:	/DVWA/login.php

Issue description

If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.

Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defense is normally ineffective and can usually be circumvented by a skilled attacker.

You should determine whether any functions accessible within frameable pages can be used by application users to perform any sensitive actions within the application.

Issue remediation

To effectively prevent framing attacks, the application should return a response header with the name **X-Frame-Options** and the value **DENY** to prevent framing altogether, or the value **SAMEORIGIN** to allow framing only by pages on the same origin as the response itself. Note that the SAMEORIGIN header can be partially bypassed if the application itself can be made to frame untrusted websites.

References

- [Web Security Academy: Clickjacking](#)
- [X-Frame-Options](#)

Vulnerability classifications

- [CWE-693: Protection Mechanism Failure](#)
- [CAPEC-103: Clickjacking](#)

Request 1

```
GET /DVWA/login.php HTTP/1.1
Host: localhost
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.5615.50 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

Response 1

```
HTTP/1.1 200 OK
Date: Fri, 19 Apr 2024 04:05:15 GMT
Server: Apache/2.4.46 (Debian)
Set-Cookie: security=impossible; path=/; HttpOnly
Set-Cookie: PHPSESSID=uuvsjct7tr7o1pfa22o5trvsdr; expires=Sat, 20-Apr-2024 04:05:15 GMT; Max-Age=86400; path=/; domain=localhost; HttpOnly; SameSite=1
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Vary: Accept-Encoding
Content-Length: 1415
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>

<html lang="en-GB">

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

    <title>Login :: Damn Vulnerable Web Application (DVWA) v1.10 *Developm
...[SNIP]...
```