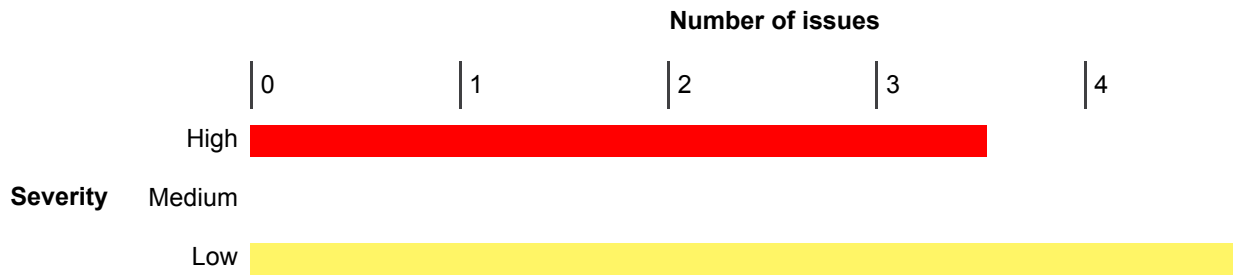
SiteWALL Web Security Assessment Report

Application: http://www.example.com

Report generated at Fri Aug 21 10:30:12 IST 2020.

Summary

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. Cross-site scripting (reflected)

1.1. http://www.example.com/contact-php-programmer.php [cemail parameter]
1.2. http://www.example.com/contact-php-programmer.php [cname parameter]
1.3. http://www.example.com/contact-php-programmer.php [projectdetails parameter]

2. Client-side JSON injection (DOM-based)

2.1. http://www.example.com/recommendations.php
2.2. http://www.example.com/recommendations.php
2.3. http://www.example.com/recommendations.php
2.4. http://www.example.com/recommendations.php

3. Path-relative style sheet import

3.1. http://www.example.com/
3.2. http://www.example.com/debug-php.php
3.3. http://www.example.com/freelance-php-mysql-web-developer-in-zurich-switzerland

---

1. Cross-site scripting (reflected)

There are 3 instances of this issue:

- /contact-php-programmer.php [cemail parameter]
- /contact-php-programmer.php [cname parameter]
- /contact-php-programmer.php [projectdetails parameter]

Issue background

Reflected cross-site scripting vulnerabilities arise when data is copied from a request and echoed into the application's immediate response in an unsafe way. An attacker can use the vulnerability to construct a request that, if issued by another application user, will cause JavaScript code supplied by the attacker to execute within the user's browser in the context of that user's session with the application.

The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.

Users can be induced to issue the attacker's crafted request in various ways. For example, the attacker can send a victim a link containing a malicious URL in an email or instant message. They can submit the link to popular web sites that allow content authoring, for example in blog comments. And they can create an innocuous looking web site that causes anyone viewing it to make arbitrary cross-domain requests to the vulnerable application (using either the GET or the POST method).

The security impact of cross-site scripting vulnerabilities is dependent upon the nature of the vulnerable application, the kinds of data and functionality that it contains, and the other applications that belong to the same domain and organization. If the application is used only to display non-sensitive public content, with no authentication or access control functionality, then a cross-site scripting flaw may be considered low risk. However, if the same application resides on a domain that can access cookies for other more security-critical applications, then the vulnerability could be used to attack those other applications, and so may be considered high risk. Similarly, if the organization that owns the application is a likely target for phishing attacks, then the vulnerability could be leveraged to lend credibility to such attacks, by injecting Trojan functionality into the vulnerable application and exploiting users' trust in the organization in order to capture credentials for other applications that it owns. In many kinds of application, such as those providing online banking functionality, cross-site scripting should always be considered high risk.

Issue remediation

In most situations where user-controllable data is copied into application responses, cross-site scripting attacks can be prevented using two layers of defenses:

- Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized.
- User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (< > etc).

In cases where the application's functionality allows users to author content using a restricted subset of HTML tags and attributes (for example, blog comments which allow limited formatting and linking), it is necessary to parse the supplied HTML to validate that it does not use any dangerous syntax; this is a non-trivial task.

Vulnerability classifications

- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)
- CWE-116: Improper Encoding or Escaping of Output
- CWE-159: Failure to Sanitize Special Element

1.1. http://www.example.com/contact-php-programmer.php [cemail parameter]

Summary

Severity:	**High**
Confidence:	**Certain**

Host: **http://www.example.com**

Path: **/contact-php-programmer.php**

Issue detail

The value of the **cemail** request parameter is copied into the value of an HTML tag attribute which is encapsulated in double quotation marks. The payload **ab9dw"><script>alert(1)</script>bh9dnxopvlf** was submitted in the cemail parameter. This input was echoed unmodified in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Request

```
GET /contact-php-programmer.php?ch=5&cname=test&cemail=homeshj@gmail.comab9dw%22%3e
%3cscript%3ealert(1)%3c%2fscript%3ebh9dnxopvlf&projectdetails=test&corr-ans=13&captcha-ans=13&
submit=Send+Message HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Origin: http://www.example.com
Connection: close
Referer: http://www.example.com/contact-php-programmer.php?ch=5
Cookie: __atuvc=1%7C32%2C5%7C33; __utma=191940543.894036465.1596872406.1596872406.1597474849.2;
__utmz=191940543.1596872406.1.1.utmcsr=103.83.192.109:2087|utmccn=(referral)|utmcmd=referral|utmcct=/;
__atuvs=5f37882075b70a3e004; __utmb=191940543.5.10.1597474849; __utmc=191940543; __utmt=1
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:38:33 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 20123

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-t
...[SNIP]...
w: rgba(0,0,0, 0.5) 0px 0px 8px; -moz-box-shadow: rgba(0,0,0, 0.5) 0px 0px 8px; -webkit-box-shadow: rgba(0,0,0,
0.5) 0px 0px 8px;height:30px;width:500px;border:1px solid grey;" value="homeshj@gmail.comab9dw">
<script>alert(1)</script>bh9dnxopvlf">
...[SNIP]...
```

1.2. http://www.example.com/contact-php-programmer.php [cname parameter]

Summary

Severity: **High**

Confidence: **Certain**

Host: **http://www.example.com**

Path: **/contact-php-programmer.php**

Issue detail

The value of the **cname** request parameter is copied into the value of an HTML tag attribute which is encapsulated in double quotation marks. The payload **hxvps"><script>alert(1)</script>cakd5ihmmn4** was submitted in the cname parameter. This input was echoed unmodified in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Request

```
GET /contact-php-programmer.php?ch=5&cname=testhxvps%22%3e%3cscript%3ealert(1)
%3c%2fscript%3ecakd5ihmmn4&cemail=homeshj@gmail.com&projectdetails=test&corr-ans=13&captcha-ans=13&
submit=Send+Message HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Origin: http://www.example.com
Connection: close
Referer: http://www.example.com/contact-php-programmer.php?ch=5
Cookie: __atuvc=1%7C32%2C5%7C33; __utma=191940543.894036465.1596872406.1596872406.1597474849.2;
__utmz=191940543.1596872406.1.1.utmcsr=103.83.192.109:2087|utmccn=(referral)|utmcmd=referral|utmcct=/;
__atuvs=5f37882075b70a3e004; __utmb=191940543.5.10.1597474849; __utmc=191940543; __utmt=1
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:37:33 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 20046

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-t
...[SNIP]...
;-moz-border-radius:5px;-webkit-border-radius:5px;box-shadow: rgba(0,0,0, 0.5) 0px 0px 8px; -moz-box-shadow:
rgba(0,0,0, 0.5) 0px 0px 8px; -webkit-box-shadow: rgba(0,0,0, 0.5) 0px 0px 8px;" value="testhxvps"><script>alert(1)
</script>cakd5ihmmn4">
...[SNIP]...
```

1.3. http://www.example.com/contact-php-programmer.php [projectdetails parameter]

Summary

Severity: **High**

Confidence: **Certain**

Host: **http://www.example.com**

Path: **/contact-php-programmer.php**

Issue detail

The value of the **projectdetails** request parameter is copied into the HTML document as plain text between tags. The payload **ytsxs<script>alert(1)</script>gr4q2lkpmdn** was submitted in the projectdetails parameter. This input was echoed unmodified in the application's response.

This proof-of-concept attack demonstrates that it is possible to inject arbitrary JavaScript into the application's response.

The original request used the POST method, however it was possible to convert the request to use the GET method, to enable easier demonstration and delivery of the attack.

Request

```
GET /contact-php-programmer.php?ch=5&cname=test&cemail=homeshj@gmail.com&
projectdetails=testytsxs%3cscript%3ealert(1)%3c%2fscript%3egr4q2lkpmdn&corr-ans=13&captcha-ans=13&
submit=Send+Message HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Origin: http://www.example.com
Connection: close
Referer: http://www.example.com/contact-php-programmer.php?ch=5
Cookie: __atuvc=1%7C32%2C5%7C33; __utma=191940543.894036465.1596872406.1596872406.1597474849.2;
__utmz=191940543.1596872406.1.1.utmcsr=103.83.192.109:2087|utmccn=(referral)|utmcmd=referral|utmcct=/;
__atuvs=5f37882075b70a3e004; __utmb=191940543.5.10.1597474849; __utmc=191940543; __utmt=1
Upgrade-Insecure-Requests: 1
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:39:13 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19848

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-t
...[SNIP]...
der-radius:5px;-webkit-border-radius:5px;box-shadow: rgba(0,0,0, 0.5) 0px 0px 8px; -moz-box-shadow: rgba(0,0,0,
0.5) 0px 0px 8px; -webkit-box-shadow: rgba(0,0,0, 0.5) 0px 0px 8px;" cols=80 rows=10>testytsxs<script>alert(1)
</script>gr4q2lkpmdn</textarea>
...[SNIP]...
```

---

2. Client-side JSON injection (DOM-based)

There are 4 instances of this issue:

- /recommendations.php
- /recommendations.php
- /recommendations.php
- /recommendations.php

Issue background

DOM-based vulnerabilities arise when a client-side script reads data from a controllable part of the DOM (for example, the URL) and processes this data in an unsafe way.

DOM-based JSON injection arises when a script incorporates controllable data into a string that is parsed as a JSON data structure and then processed by the application. An attacker may be able to use this behavior to construct a URL that, if visited by another application user, will cause arbitrary JSON data to be processed. Depending on the purpose for which this data is used, it may be possible to subvert the application's logic, or cause unintended actions on behalf of the user.

SiteWALL automatically identifies this issue using static code analysis, which may lead to false positives that are not actually exploitable. The relevant code and execution paths should be reviewed to determine whether this vulnerability is indeed present, or whether mitigations are in place that would prevent exploitation.

Issue remediation

The most effective way to avoid DOM-based JSON injection vulnerabilities is not to parse as JSON any string containing data that originated from an untrusted source. If the desired functionality of the application means that this behavior is unavoidable, then defenses must be implemented within the client-side code to prevent malicious data from modifying the JSON structure in inappropriate ways. This may involve strict validation of specific items to ensure they do not contain any characters that may interfere with the structure of the JSON when it is parsed.

Vulnerability classifications

- CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
- CWE-116: Improper Encoding or Escaping of Output
- CWE-159: Failure to Sanitize Special Element

2.1. http://www.example.com/recommendations.php

Summary

Severity:	**Low**
Confidence:	**Firm**
Host:	**http://www.example.com**
Path:	**/recommendations.php**

Issue detail

The application may be vulnerable to DOM-based client-side JSON injection. Data is read from **window.name** and passed to **JSON.parse**.

Note: The name of the current window is a valid attack vector for DOM-based vulnerabilities. An attacker can directly control the name of the targeted application's window by using code on their own domain to load the targeted page using either window.open() or an iframe tag, and specifying the desired window name.

Request

```
GET /recommendations.php?ch=6 HTTP/1.1
Host: www.example.com
```

```
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19579

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
```

Dynamic analysis

Data is read from **window.name** and passed to **JSON.parse**.

The following value was injected into the source and reached the sink without any modification:

```
xzq7ybxbrk%2527%2522`'"/xzq7ybxbrk/><xzq7ybxbrk/\>jb8kswpv7b&
```

The stack trace at the source was:

```
at Object.AXwjI (<anonymous>:1:336094)
at Object.ozYtQ (<anonymous>:1:753291)
at get (<anonymous>:1:755357)
at M (http://platform.linkedin.com/in.js:18:93084)
at new t (http://platform.linkedin.com/in.js:18:109222)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:111689)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The stack trace at the sink was:

```
at JSON.<anonymous> (<anonymous>:1:851609)
at M (http://platform.linkedin.com/in.js:18:93071)
at new t (http://platform.linkedin.com/in.js:18:109222)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:111689)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The following proof of concept was generated for this issue:

```
<a href="http://www.example.com/recommendations.php?ch=6" target="">PoC</a>
```

2.2. http://www.example.com/recommendations.php

Summary

Severity:	**Low**
Confidence:	**Firm**
Host:	**http://www.example.com**
Path:	**/recommendations.php**

Issue detail

The application may be vulnerable to DOM-based client-side JSON injection. Data is read from **window.name** and passed to **JSON.parse**.

Note: The name of the current window is a valid attack vector for DOM-based vulnerabilities. An attacker can directly control the name of the targeted application's window by using code on their own domain to load the targeted page using either window.open() or an iframe tag, and specifying the desired window name.

Request

```
GET /recommendations.php?ch=6 HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
```

Response

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19579

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
```

Dynamic analysis

Data is read from **window.name** and passed to **JSON.parse**.

The following value was injected into the source and reached the sink without any modification:

```
tujpyttdjs%2527%2522`'"/tujpyttdjs/><tujpyttdjs/\>yt2t4by6gp&
```

The stack trace at the source was:

```
at Object.AXwjI (<anonymous>:1:336094)
at Object.ozYtQ (<anonymous>:1:753291)
at get (<anonymous>:1:755357)
at M (http://platform.linkedin.com/in.js:18:93084)
at http://platform.linkedin.com/in.js:18:93926
at new e (http://platform.linkedin.com/in.js:18:93937)
at new t (http://platform.linkedin.com/in.js:18:107852)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:111689)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The stack trace at the sink was:

```
at JSON.<anonymous> (<anonymous>:1:851609)
at M (http://platform.linkedin.com/in.js:18:93071)
at http://platform.linkedin.com/in.js:18:93926
at new e (http://platform.linkedin.com/in.js:18:93937)
at new t (http://platform.linkedin.com/in.js:18:107852)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:111689)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The following proof of concept was generated for this issue:

```
<a href="http://www.example.com/recommendations.php?ch=6" target="">PoC</a>
```

2.3. http://www.example.com/recommendations.php

Summary

Severity:	**Low**
Confidence:	**Firm**
Host:	**http://www.example.com**
Path:	**/recommendations.php**

Issue detail

The application may be vulnerable to DOM-based client-side JSON injection. Data is read from **window.name** and passed to **JSON.parse()**.

Note: The name of the current window is a valid attack vector for DOM-based vulnerabilities. An attacker can directly control the name of the targeted application's window by using code on their own domain to load the targeted page using either window.open() or an iframe tag, and specifying the desired window name.

Request 1

```
GET /recommendations.php?ch=6 HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
```

Response 1

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19579

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
</script>
<script src="//platform.linkedin.com/in.js" type="text/javascript"></script>
...[SNIP]...
```

Request 2

```
GET /in.js HTTP/1.1
Host: platform.linkedin.com
Accept: */*
Accept-Language: en-US,en-GB;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/84.0.4147.105 Safari/537.36
Connection: close
Cache-Control: max-age=0
Accept-Encoding: gzip, deflate
```

Response 2

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 3066
Cache-Control: public, max-age=3600
Content-Type: text/javascript; charset=UTF-8
Date: Sat, 15 Aug 2020 14:38:22 GMT
Expires: Sat, 15 Aug 2020 14:47:16 GMT
Last-Modified: Sat, 15 Aug 2020 13:47:16 GMT
Server: ECAcc (nag/9958)
Vary: Accept-Encoding
X-Cache: HIT
X-CDN: ECST
X-CDN-CLIENT-IP-VERSION: IPV4
X-CDN-Proto: HTTP1
X-Li-Fabric: prod-lor1
X-Li-Pop: prod-tmu1
X-LI-Proto: http/1.1
X-LI-UUID: iFGnhRd1KxagBBMZbisAAA==
```

```
Content-Length: 185763
Connection: close

/* xdoor-frontend: v0.1.180 (Mon, 27 Jul 2020 17:05:34 GMT) */
(function() {

var PAYLOAD = {"ENV":{"widget":{"alumni_url":"https://www.linkedin.com/cws/alumni","followmember_url":"https:
//www.linkedi
...[SNIP]...
0),k=n.n(x),_=n(4),S=n.n(_),I=n(189),E=n.n(I);function O(t){return t.trim().toLowerCase().replace(/[^a-zA-
Z0-9]/g,"")}var N=n(65),A=n(33),C=n.n(A),j=n(66),P=n(7),T=n.n(P);function M(){var t=void
0;try{t=JSON.parse(window.name)}catch(t){}return t&&"object"===(void 0===t?"undefined":T()(t))?t:{}}var
L={lang:"en_US",init:!0,parse:!0,compat:!0,dataNamespace:void 0,onLoad:void 0,templateMarkers:"<?js ?>
...[SNIP]...
```

Static analysis

Data is read from **window.name** and passed to **JSON.parse()** via the following statement:

- `t=JSON.parse(window.name)`

Dynamic analysis

Data is read from **window.name** and passed to **JSON.parse**.

The following value was injected into the source and reached the sink without any modification:

```
ahgk4f79uo%2527%2522`'"/ahgk4f79uo/><ahgk4f79uo/\>ag4dsg51lu&
```

The stack trace at the source was:

```
at Object.AXwjI (<anonymous>:1:336094)
at Object.ozYtQ (<anonymous>:1:753291)
at get (<anonymous>:1:755357)
at M (http://platform.linkedin.com/in.js:18:93084)
at http://platform.linkedin.com/in.js:18:93926
at new e (http://platform.linkedin.com/in.js:18:93937)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:105318)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The stack trace at the sink was:

```
at JSON.<anonymous> (<anonymous>:1:851609)
at M (http://platform.linkedin.com/in.js:18:93071)
at http://platform.linkedin.com/in.js:18:93926
at new e (http://platform.linkedin.com/in.js:18:93937)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:105318)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The following proof of concept was generated for this issue:

```
<a href="http://www.example.com/recommendations.php?ch=6" target="">PoC</a>
```

2.4. http://www.example.com/recommendations.php

Summary

Severity:	**Low**
Confidence:	**Firm**
Host:	**http://www.example.com**
Path:	**/recommendations.php**

Issue detail

The application may be vulnerable to DOM-based client-side JSON injection. Data is read from **window.name** and passed to **JSON.parse()**.

Note: The name of the current window is a valid attack vector for DOM-based vulnerabilities. An attacker can directly control the name of the targeted application's window by using code on their own domain to load the targeted page using either window.open() or an iframe tag, and specifying the desired window name.

Request 1

```
GET /recommendations.php?ch=6 HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
```

Response 1

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:22:44 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19579

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
</script>
<script src="//platform.linkedin.com/in.js" type="text/javascript"></script>
...[SNIP]...
```

Request 2

```
GET /in.js HTTP/1.1
Host: platform.linkedin.com
Accept: */*
Accept-Language: en-US,en-GB;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/84.0.4147.105 Safari/537.36
Connection: close
Cache-Control: max-age=0
Accept-Encoding: gzip, deflate
```

Response 2

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Age: 3066
Cache-Control: public, max-age=3600
Content-Type: text/javascript; charset=UTF-8
Date: Sat, 15 Aug 2020 14:38:22 GMT
Expires: Sat, 15 Aug 2020 14:47:16 GMT
Last-Modified: Sat, 15 Aug 2020 13:47:16 GMT
Server: ECAcc (nag/9958)
Vary: Accept-Encoding
X-Cache: HIT
X-CDN: ECST
X-CDN-CLIENT-IP-VERSION: IPV4
X-CDN-Proto: HTTP1
X-Li-Fabric: prod-lor1
X-Li-Pop: prod-tmu1
X-LI-Proto: http/1.1
X-LI-UUID: iFGnhRd1KxagBBMZbisAAA==
Content-Length: 185763
Connection: close

/* xdoor-frontend: v0.1.180 (Mon, 27 Jul 2020 17:05:34 GMT) */
(function() {

var PAYLOAD = {"ENV":{"widget":{"alumni_url":"https://www.linkedin.com/cws/alumni","followmember_url":"https:
//www.linkedi
...[SNIP]...
0),k=n.n(x),_=n(4),S=n.n(_),I=n(189),E=n.n(I);function O(t){return t.trim().toLowerCase().replace(/[^a-zA-
Z0-9]/g,"")}var N=n(65),A=n(33),C=n.n(A),j=n(66),P=n(7),T=n.n(P);function M(){var t=void
0;try{t=JSON.parse(window.name)}catch(t){}return t&&"object"===(void 0===t?"undefined":T()(t))?t:{}}var
L={lang:"en_US",init:!0,parse:!0,compat:!0,dataNamespace:void 0,onLoad:void 0,templateMarkers:"<?js ?>
...[SNIP]...
```

Static analysis

Data is read from **window.name** and passed to **JSON.parse()** via the following statement:

- `t=JSON.parse(window.name)`

Dynamic analysis

Data is read from **window.name** and passed to **JSON.parse**.

The following value was injected into the source and reached the sink without any modification:

```
ahgk4f79uo%2527%2522`'"/ahgk4f79uo/><ahgk4f79uo/\>ag4dsg51lu&
```

The stack trace at the source was:

```
at Object.AXwjI (<anonymous>:1:336094)
at Object.ozYtQ (<anonymous>:1:753291)
at get (<anonymous>:1:755357)
at M (http://platform.linkedin.com/in.js:18:93084)
at http://platform.linkedin.com/in.js:18:93926
at new e (http://platform.linkedin.com/in.js:18:93937)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:105318)
at n (http://platform.linkedin.com/in.js:7:5462)
```

```
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The stack trace at the sink was:

```
at JSON.<anonymous> (<anonymous>:1:851609)
at M (http://platform.linkedin.com/in.js:18:93071)
at http://platform.linkedin.com/in.js:18:93926
at new e (http://platform.linkedin.com/in.js:18:93937)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:105318)
at n (http://platform.linkedin.com/in.js:7:5462)
at http://platform.linkedin.com/in.js:7:6262
at Object.<anonymous> (http://platform.linkedin.com/in.js:7:6273)
at n (http://platform.linkedin.com/in.js:7:110)
at Module.<anonymous> (http://platform.linkedin.com/in.js:18:112546)
at n (http://platform.linkedin.com/in.js:7:110)
at http://platform.linkedin.com/in.js:7:903
at http://platform.linkedin.com/in.js:7:913
at http://platform.linkedin.com/in.js:19:2
```

The following proof of concept was generated for this issue:

```
<a href="http://www.example.com/recommendations.php?ch=6" target="">PoC</a>
```

---

3. Path-relative style sheet import

There are 3 instances of this issue:

- /
- /debug-php.php
- /freelance-php-mysql-web-developer-in-zurich-switzerland

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.

Vulnerability classifications

- CWE-16: Configuration

3.1. http://www.example.com/

Summary

Severity:	**Information**
Confidence:	**Tentative**
Host:	**http://www.example.com**
Path:	**/**

Issue detail

The application may be vulnerable to path-relative style sheet import (PRSSI) attacks. The response contains a path-relative style sheet import, and so condition 1 for an exploitable vulnerability is present (see issue background). The response can also 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.) This means that condition 3 for an exploitable vulnerability is probably present if condition 2 is present.

SiteWALL was not able to confirm that the other conditions hold, and you should manually investigate this issue to confirm whether they do hold.

Request 1

```
GET / HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.11; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: __atuvc=1%7C32%2C1%7C33; __utma=191940543.894036465.1596872406.1596872406.1597474849.2;
__utmz=191940543.1596872406.1.1.utmcsr=103.83.192.109:2087|utmccn=(referral)|utmcmd=referral|utmcct=/;
__atuvs=5f37882075b70a3e000; __utmb=191940543.1.10.1597474849; __utmc=191940543; __utmt=1
Upgrade-Insecure-Requests: 1
```

```
Cache-Control: max-age=0
```

Response 1

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 07:02:15 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 20926

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
<meta name="keywords" content="php, mysql, web development, prestashop, ecommerce, online php mysql development, freelance programmer" />
<link href="styles.css" rel="stylesheet" type="text/css" media="screen" />


<!-- Google Analytics Content Experiment code -->
...[SNIP]...
<!-- for facebook -->
<link href="navi.css" rel="stylesheet" type="text/css" />
<!--
<br>
...[SNIP]...
```

3.2. http://www.example.com/debug-php.php

Summary

Severity:	**Information**
Confidence:	**Firm**
Host:	**http://www.example.com**
Path:	**/debug-php.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.

Request 1

```
GET /debug-php.php HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
```

Response 1

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:17:42 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 20084

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
web programming, freelance programing, freelance web programing,web freelancer, free lance, web
devloper,website programmer, web site programmer, web site programming, website programming" />
<link href="styles.css" rel="stylesheet" type="text/css" media="screen" />
<script language="JavaScript" src="gen_validatorv2.js" type="text/javascript">
...[SNIP]...
```

Request 2

```
GET /debug-php.php/lqn9ce/ HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
Cookie: __utma=191940543.894036465.1596872406.1596872406.1597474849.2;
__utmz=191940543.1596872406.1.1.utmcsr=103.83.192.109:2087|utmccn=(referral)|utmcmd=referral|utmcct=/;
__utmc=191940543; __utmt=1; __atuvc=1%7C32%2C11%7C33; __atuvs=5f37882075b70a3e00a;
__utmb=191940543.11.10.1597474849
```

Response 2

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:43:21 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19909

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
web programming, freelance programing, freelance web programing,web freelancer, free lance, web
devloper,website programmer, web site programmer, web site programming, website programming" />
```

```
<link href="styles.css" rel="stylesheet" type="text/css" media="screen" />
<script language="JavaScript" src="gen_validatorv2.js" type="text/javascript">
...[SNIP]...
```

Request 3

```
GET /debug-php.php/lqn9ce/styles.css HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
Cookie: __utma=191940543.894036465.1596872406.1596872406.1597474849.2;
__utmz=191940543.1596872406.1.1.utmcsr=103.83.192.109:2087|utmccn=(referral)|utmcmd=referral|utmcct=/;
__utmc=191940543; __utmt=1; __atuvc=1%7C32%2C11%7C33; __atuvs=5f37882075b70a3e00a;
__utmb=191940543.11.10.1597474849
```

Response 3

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:43:21 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 19983

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="conten
...[SNIP]...
```

3.3. http://www.example.com/freelance-php-mysql-web-developer-in-zurich-switzerland

Summary

Severity:	**Information**
Confidence:	**Tentative**
Host:	**http://www.example.com**
Path:	**/freelance-php-mysql-web-developer-in-zurich-switzerland**

Issue detail

The application may be vulnerable to path-relative style sheet import (PRSSI) attacks. The response contains a path-relative style sheet import, and so condition 1 for an exploitable vulnerability is present (see issue background). The response can also 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.) This means that condition 3 for an exploitable vulnerability is probably present if condition 2 is present.

SiteWALL was not able to confirm that the other conditions hold, and you should manually investigate this issue to confirm whether they do hold.

Request 1

```
GET /freelance-php-mysql-web-developer-in-zurich-switzerland HTTP/1.1
Host: www.example.com
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/84.0.4147.105 Safari/537.36
Connection: close
```

Response 1

```
HTTP/1.1 200 OK
Date: Sat, 15 Aug 2020 14:17:42 GMT
Server: Apache
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Length: 28080

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="content-t
...[SNIP]...
meta name="keywords" content="Freelance Php Mysql Web Developer In Zurich Switzerland find a web developer,
professional web developer, web programming quote, freelance programmer, web programmer" />
<link href="styles.css" rel="stylesheet" type="text/css" media="screen" />
<script language="JavaScript" src="gen_validatorv2.js" type="text/javascript">
...[SNIP]...
<!-- for facebook -->
<link href="navi.css" rel="stylesheet" type="text/css" />
<!--
<br>
...[SNIP]...
```