# ZAP Scanning Report

### Summary of Alerts

### Generated on Mon, 23 Aug 2021 18:57:16

| **Risk Level** | **Number of Alerts** |
| --- | --- |
| [High](#gjdgxs) | 6 |
| [Medium](#30j0zll) | 3 |
| [Low](#1fob9te) | 5 |
| [Informational](#3znysh7) | 2 |

### Alerts

| **Name** | **Risk Level** | **Number of Instances** |
| --- | --- | --- |
| Cross Site Scripting (DOM Based) | High | 23 |
| Cross Site Scripting (Persistent) | High | 1 |
| Cross Site Scripting (Reflected) | High | 3 |
| Remote OS Command Injection | High | 2 |
| SQL Injection | High | 2 |
| Buffer Overflow | Medium | 2 |
| Directory Browsing | Medium | 11 |
| X-Frame-Options Header Not Set | Medium | 20 |
| Absence of Anti-CSRF Tokens | Low | 26 |
| Cookie No HttpOnly Flag | Low | 1 |
| Cookie without SameSite Attribute | Low | 1 |
| Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) | Low | 35 |
| X-Content-Type-Options Header Missing | Low | 33 |
| Information Disclosure - Sensitive Information in URL | Informational | 1 |
| Timestamp Disclosure - Unix | Informational | 5 |

### Alert Detail

| **High (Medium)** | **Cross Site Scripting (DOM Based)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.  When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.  There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.  Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.  Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/home.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/passcheck.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/calendar.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/cart/action.php?action=delete#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/tos.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/cart/review.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=14#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/guestbook.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/register.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures/upload.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/passcheck.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/logout.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | GET |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| URL | http://10.1.2.6/WackoPicko/users/login.php#jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Method | POST |
| Attack | #jaVasCript:/\*-/\*`/\*\`/\*'/\*"/\*\*/(/\* \*/oNcliCk=alert() )//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>\x3csVg/<sVg/oNloAd=alert()//>\x3e |
| Instances | 23 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.  Phases: Implementation; Architecture and Design  Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.  For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.  Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.  Phase: Architecture and Design  For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.  If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.  Phase: Implementation  For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.  To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.  Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Other information | Tag name: div Att name: null Att id: |
|  | |
| Reference | http://projects.webappsec.org/Cross-Site-Scripting  http://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

| **High (Medium)** | **Cross Site Scripting (Persistent)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.  When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.  There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.  Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.  Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Parameter | comment |
| Attack | <script>alert(1);</script> |
| Instances | 1 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.  Phases: Implementation; Architecture and Design  Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.  For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.  Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.  Phase: Architecture and Design  For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.  If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.  Phase: Implementation  For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.  To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.  Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Other information | Source URL: http://10.1.2.6/WackoPicko/guestbook.php |
|  | |
| Reference | http://projects.webappsec.org/Cross-Site-Scripting  http://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

| **High (Medium)** | **Cross Site Scripting (Reflected)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.  When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.  There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.  Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.  Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=%22%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E |
| Method | GET |
| Parameter | query |
| Attack | "><script>alert(1);</script> |
| Evidence | "><script>alert(1);</script> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Parameter | comment |
| Attack | <script>alert(1);</script> |
| Evidence | <script>alert(1);</script> |
| Instances | 2 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.  Phases: Implementation; Architecture and Design  Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.  For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.  Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.  Phase: Architecture and Design  For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.  If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.  Phase: Implementation  For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.  To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.  Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Reference | http://projects.webappsec.org/Cross-Site-Scripting  http://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

| **High (Low)** | **Cross Site Scripting (Reflected)** |
| --- | --- |
| Description | Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.  When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.  There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.  Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.  Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code. |
|  | |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | username |
| Attack | '"<script>alert(1);</script> |
| Evidence | '"<script>alert(1);</script> |
| Instances | 1 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.  Phases: Implementation; Architecture and Design  Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.  For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.  Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.  Phase: Architecture and Design  For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.  If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.  Phase: Implementation  For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.  To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.  Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere. |
| Reference | http://projects.webappsec.org/Cross-Site-Scripting  http://cwe.mitre.org/data/definitions/79.html |
| CWE Id | 79 |
| WASC Id | 8 |
| Source ID | 1 |

| **High (Medium)** | **Remote OS Command Injection** |
| --- | --- |
| Description | Attack technique used for unauthorized execution of operating system commands. This attack is possible when an application accepts untrusted input to build operating system commands in an insecure manner involving improper data sanitization, and/or improper calling of external programs. |
|  | |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Parameter | password |
| Attack | ZAP&sleep 15& |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Parameter | name |
| Attack | ZAP';sleep 15;' |
| Instances | 2 |
| Solution | If at all possible, use library calls rather than external processes to recreate the desired functionality.  Run your code in a "jail" or similar sandbox environment that enforces strict boundaries between the process and the operating system. This may effectively restrict which files can be accessed in a particular directory or which commands can be executed by your software.  OS-level examples include the Unix chroot jail, AppArmor, and SELinux. In general, managed code may provide some protection. For example, java.io.FilePermission in the Java SecurityManager allows you to specify restrictions on file operations.  This may not be a feasible solution, and it only limits the impact to the operating system; the rest of your application may still be subject to compromise.  For any data that will be used to generate a command to be executed, keep as much of that data out of external control as possible. For example, in web applications, this may require storing the command locally in the session's state instead of sending it out to the client in a hidden form field.  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  For example, consider using the ESAPI Encoding control or a similar tool, library, or framework. These will help the programmer encode outputs in a manner less prone to error.  If you need to use dynamically-generated query strings or commands in spite of the risk, properly quote arguments and escape any special characters within those arguments. The most conservative approach is to escape or filter all characters that do not pass an extremely strict allow list (such as everything that is not alphanumeric or white space). If some special characters are still needed, such as white space, wrap each argument in quotes after the escaping/filtering step. Be careful of argument injection.  If the program to be executed allows arguments to be specified within an input file or from standard input, then consider using that mode to pass arguments instead of the command line.  If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.  Some languages offer multiple functions that can be used to invoke commands. Where possible, identify any function that invokes a command shell using a single string, and replace it with a function that requires individual arguments. These functions typically perform appropriate quoting and filtering of arguments. For example, in C, the system() function accepts a string that contains the entire command to be executed, whereas execl(), execve(), and others require an array of strings, one for each argument. In Windows, CreateProcess() only accepts one command at a time. In Perl, if system() is provided with an array of arguments, then it will quote each of the arguments.  Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.  When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."  When constructing OS command strings, use stringent allow lists that limit the character set based on the expected value of the parameter in the request. This will indirectly limit the scope of an attack, but this technique is less important than proper output encoding and escaping.  Note that proper output encoding, escaping, and quoting is the most effective solution for preventing OS command injection, although input validation may provide some defense-in-depth. This is because it effectively limits what will appear in output. Input validation will not always prevent OS command injection, especially if you are required to support free-form text fields that could contain arbitrary characters. For example, when invoking a mail program, you might need to allow the subject field to contain otherwise-dangerous inputs like ";" and ">" characters, which would need to be escaped or otherwise handled. In this case, stripping the character might reduce the risk of OS command injection, but it would produce incorrect behavior because the subject field would not be recorded as the user intended. This might seem to be a minor inconvenience, but it could be more important when the program relies on well-structured subject lines in order to pass messages to other components.  Even if you make a mistake in your validation (such as forgetting one out of 100 input fields), appropriate encoding is still likely to protect you from injection-based attacks. As long as it is not done in isolation, input validation is still a useful technique, since it may significantly reduce your attack surface, allow you to detect some attacks, and provide other security benefits that proper encoding does not address. |
| Reference | http://cwe.mitre.org/data/definitions/78.html  https://owasp.org/www-community/attacks/Command\_Injection |
| CWE Id | 78 |
| WASC Id | 31 |
| Source ID | 1 |

| **High (Medium)** | **SQL Injection** |
| --- | --- |
| Description | SQL injection may be possible. |
|  | |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=14+AND+1%3D1+--+ |
| Method | GET |
| Parameter | picid |
| Attack | 14 AND 1=1 -- |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | username |
| Attack | ZAP' AND '1'='1' -- |
| Instances | 2 |
| Solution | Do not trust client side input, even if there is client side validation in place.  In general, type check all data on the server side.  If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'  If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.  If database Stored Procedures can be used, use them.  Do \*not\* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!  Do not create dynamic SQL queries using simple string concatenation.  Escape all data received from the client.  Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input.  Apply the principle of least privilege by using the least privileged database user possible.  In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.  Grant the minimum database access that is necessary for the application. |
| Other information | The page results were successfully manipulated using the boolean conditions [14 AND 1=1 -- ] and [14 AND 1=2 -- ]  The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison  Data was returned for the original parameter.  The vulnerability was detected by successfully restricting the data originally returned, by manipulating the parameter |
|  | |
| Reference | https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html |
| CWE Id | 89 |
| WASC Id | 19 |
| Source ID | 1 |

| **Medium (Medium)** | **Buffer Overflow** |
| --- | --- |
| Description | Buffer overflow errors are characterized by the overwriting of memory spaces of the background web process, which should have never been modified intentionally or unintentionally. Overwriting values of the IP (Instruction Pointer), BP (Base Pointer) and other registers causes exceptions, segmentation faults, and other process errors to occur. Usually these errors end execution of the application in an unexpected way. |
|  | |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | GET |
| Parameter | page |
| Attack |  |
| Evidence | Connection: close |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Parameter | page |
| Attack |  |
| Evidence | Connection: close |
| Instances | 2 |
| Solution | Rewrite the background program using proper return length checking. This will require a recompile of the background executable. |
| Other information | Potential Buffer Overflow. The script closed the connection and threw a 500 Internal Server Error |
|  | |
| Reference | https://owasp.org/www-community/attacks/Buffer\_overflow\_attack |
| CWE Id | 120 |
| WASC Id | 7 |
| Source ID | 1 |

| **Medium (Medium)** | **Directory Browsing** |
| --- | --- |
| Description | It is possible to view the directory listing. Directory listing may reveal hidden scripts, include files, backup source files, etc. which can be accessed to read sensitive information. |
|  | |
| URL | http://10.1.2.6/WackoPicko/upload/flowers/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/flowers/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/css/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/css/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/waterfall/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/waterfall/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/doggie/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/doggie/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/pictures/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/pictures/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/toga/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/toga/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/cart/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/cart/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/upload/house/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/upload/house/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/users/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/users/ |
| Evidence | Parent Directory |
| URL | http://10.1.2.6/WackoPicko/css/blueprint/ |
| Method | GET |
| Attack | http://10.1.2.6/WackoPicko/css/blueprint/ |
| Evidence | Parent Directory |
| Instances | 11 |
| Solution | Disable directory browsing. If this is required, make sure the listed files does not induce risks. |
| Reference | http://httpd.apache.org/docs/mod/core.html#options  http://alamo.satlug.org/pipermail/satlug/2002-February/000053.html |
| CWE Id | 548 |
| WASC Id | 48 |
| Source ID | 1 |

| **Medium (Medium)** | **X-Frame-Options Header Not Set** |
| --- | --- |
| Description | X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks. |
|  | |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/tos.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630007620 |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | GET |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Parameter | X-Frame-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Parameter | X-Frame-Options |
| Instances | 20 |
| Solution | Most modern Web browsers support the X-Frame-Options HTTP header. Ensure it's set on all web pages returned by your site (if you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive. |
| Reference | https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options |
| CWE Id | 1021 |
| WASC Id | 15 |
| Source ID | 3 |

| **Low (Medium)** | **Absence of Anti-CSRF Tokens** |
| --- | --- |
| Description | No Anti-CSRF tokens were found in a HTML submission form.  A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user. By contrast, cross-site scripting (XSS) exploits the trust that a user has for a web site. Like XSS, CSRF attacks are not necessarily cross-site, but they can be. Cross-site request forgery is also known as CSRF, XSRF, one-click attack, session riding, confused deputy, and sea surf.  CSRF attacks are effective in a number of situations, including:  \* The victim has an active session on the target site.  \* The victim is authenticated via HTTP auth on the target site.  \* The victim is on the same local network as the target site.  CSRF has primarily been used to perform an action against a target site using the victim's privileges, but recent techniques have been discovered to disclose information by gaining access to the response. The risk of information disclosure is dramatically increased when the target site is vulnerable to XSS, because XSS can be used as a platform for CSRF, allowing the attack to operate within the bounds of the same-origin policy. |
|  | |
| URL | http://10.1.2.6/WackoPicko/calendar.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/guestbook.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1 |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/guestbook.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/passcheck.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/passcheck.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/users/register.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Evidence | <form action="/WackoPicko/users/login.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/users/login.php" method="POST"> |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Evidence | <form action="/WackoPicko/pictures/search.php" method="get" style="display:inline;"> |
| Instances | 26 |
| Solution | Phase: Architecture and Design  Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  For example, use anti-CSRF packages such as the OWASP CSRFGuard.  Phase: Implementation  Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script.  Phase: Architecture and Design  Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330).  Note that this can be bypassed using XSS.  Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.  Note that this can be bypassed using XSS.  Use the ESAPI Session Management control.  This control includes a component for CSRF.  Do not use the GET method for any request that triggers a state change.  Phase: Implementation  Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons. |
| Other information | No known Anti-CSRF token [anticsrf, CSRFToken, \_\_RequestVerificationToken, csrfmiddlewaretoken, authenticity\_token, OWASP\_CSRFTOKEN, anoncsrf, csrf\_token, \_csrf, \_csrfSecret, \_\_csrf\_magic, CSRF] was found in the following HTML form: [Form 1: "query2" ]. |
|  | |
| Reference | http://projects.webappsec.org/Cross-Site-Request-Forgery  http://cwe.mitre.org/data/definitions/352.html |
| CWE Id | 352 |
| WASC Id | 9 |
| Source ID | 3 |

| **Low (Medium)** | **Cookie No HttpOnly Flag** |
| --- | --- |
| Description | A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible. |
|  | |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | PHPSESSID |
| Evidence | Set-Cookie: PHPSESSID |
| Instances | 1 |
| Solution | Ensure that the HttpOnly flag is set for all cookies. |
| Reference | https://owasp.org/www-community/HttpOnly |
| CWE Id | 1004 |
| WASC Id | 13 |
| Source ID | 3 |

| **Low (Medium)** | **Cookie without SameSite Attribute** |
| --- | --- |
| Description | A cookie has been set without the SameSite attribute, which means that the cookie can be sent as a result of a 'cross-site' request. The SameSite attribute is an effective counter measure to cross-site request forgery, cross-site script inclusion, and timing attacks. |
|  | |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | PHPSESSID |
| Evidence | Set-Cookie: PHPSESSID |
| Instances | 1 |
| Solution | Ensure that the SameSite attribute is set to either 'lax' or ideally 'strict' for all cookies. |
| Reference | https://tools.ietf.org/html/draft-ietf-httpbis-cookie-same-site |
| CWE Id | 1275 |
| WASC Id | 13 |
| Source ID | 3 |

| **Low (Medium)** | **Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s)** |
| --- | --- |
| Description | The web/application server is leaking information via one or more "X-Powered-By" HTTP response headers. Access to such information may facilitate attackers identifying other frameworks/components your web application is reliant upon and the vulnerabilities such components may be subject to. |
|  | |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/tos.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=13 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=14 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/cart/action.php?action=delete |
| Method | POST |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/upload.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=11 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/recent.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/css/stylings.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | POST |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=12 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/users/logout.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/users/register.php |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/pictures/view.php?picid=7 |
| Method | GET |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Evidence | X-Powered-By: PHP/5.3.2-1ubuntu4.30 |
| Instances | 35 |
| Solution | Ensure that your web server, application server, load balancer, etc. is configured to suppress "X-Powered-By" headers. |
| Reference | http://blogs.msdn.com/b/varunm/archive/2013/04/23/remove-unwanted-http-response-headers.aspx  http://www.troyhunt.com/2012/02/shhh-dont-let-your-response-headers.html |
| CWE Id | 200 |
| WASC Id | 13 |
| Source ID | 3 |

| **Low (Medium)** | **X-Content-Type-Options Header Missing** |
| --- | --- |
| Description | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing. |
|  | |
| URL | http://10.1.2.6/WackoPicko/upload/house/hodjjgld.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/css/blueprint/screen.css |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/ |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/guestbook.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/pictures/search.php?query=ZAP |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/house/our\_house.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/css/stylings.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/flowers/flowers.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/house/My\_House.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/toga/togas.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/toga/togasfs.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/users/login.php |
| Method | POST |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/passcheck.php |
| Method | POST |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/upload/waterfall/Waterfall.128\_128.jpg |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/admin/index.php?page=login |
| Method | POST |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Parameter | X-Content-Type-Options |
| URL | http://10.1.2.6/WackoPicko/cart/review.php |
| Method | GET |
| Parameter | X-Content-Type-Options |
| Instances | 33 |
| Solution | Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages.  If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing. |
| Other information | This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type.  At "High" threshold this scan rule will not alert on client or server error responses. |
|  | |
| Reference | http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx  https://owasp.org/www-community/Security\_Headers |
| CWE Id | 693 |
| WASC Id | 15 |
| Source ID | 3 |

| **Informational (Medium)** | **Information Disclosure - Sensitive Information in URL** |
| --- | --- |
| Description | The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment. |
|  | |
| URL | http://10.1.2.6/WackoPicko/users/sample.php?userid=1 |
| Method | GET |
| Parameter | userid |
| Evidence | userid |
| Instances | 1 |
| Solution | Do not pass sensitive information in URIs. |
| Other information | The URL contains potentially sensitive information. The following string was found via the pattern: user  userid |
|  | |
| Reference |  |
| CWE Id | 200 |
| WASC Id | 13 |
| Source ID | 3 |

| **Informational (Low)** | **Timestamp Disclosure - Unix** |
| --- | --- |
| Description | A timestamp was disclosed by the application/web server - Unix |
|  | |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630094020 |
| Method | GET |
| Evidence | 1630180420 |
| URL | http://10.1.2.6/WackoPicko/calendar.php |
| Method | GET |
| Evidence | 1629834820 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629921220 |
| Method | GET |
| Evidence | 1630007620 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1630007620 |
| Method | GET |
| Evidence | 1630094020 |
| URL | http://10.1.2.6/WackoPicko/calendar.php?date=1629834820 |
| Method | GET |
| Evidence | 1629921220 |
| Instances | 5 |
| Solution | Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns. |
| Other information | 1630180420, which evaluates to: 2021-08-28 15:53:40 |
|  | |
| Reference | http://projects.webappsec.org/w/page/13246936/Information%20Leakage |
| CWE Id | 200 |
| WASC Id | 13 |
| Source ID | 3 |