Site: http://192.168.233.128

Generated on Thu, 29 Jun 2023 13:37:04

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	9

Alerts

Name	Risk Level	Number of Instances
Cross Site Scripting (Reflected)	High	1
Hash Disclosure - MD5 Crypt	High	5
Path Traversal	High	1
Remote Code Execution - CVE-2012-1823	High	2
Source Code Disclosure - CVE-2012-1823	High	2
Absence of Anti-CSRF Tokens	Medium	46
Application Error Disclosure	Medium	21
Content Security Policy (CSP) Header Not Set	Medium	141
Directory Browsing	Medium	5
Hidden File Found	Medium	2
Missing Anti-clickjacking Header	Medium	86
Parameter Tampering	Medium	1
Vulnerable JS Library	Medium	1
XSLT Injection	Medium	1

Alert Detail

High	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://192.168.233.128/mutillidae/index.php?page=%22%3E%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E
Method	GET
Attack	"><scrIpt>alert(1);</scRipt>
Evidence	"><scrIpt>alert(1);</scRipt>
Instances	1
	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.

Solution	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

Plugin Id	40012

High	Hash Disclosure - MD5 Crypt
Description	A hash was disclosed by the web server. - MD5 Crypt
URL	http://192.168.233.128/mutillidae/?page=source-viewer.php
Method	GET
Attack	
Evidence	$1$12485267$TjSic/fv9vlo9lb2qpVrP/
URL	http://192.168.233.128/mutillidae/index.php?page=source-viewer.php
Method	GET
Attack	
Evidence	$1$86164818$tst4MkTChCospYeVZnpqa/
URL	http://192.168.233.128/mutillidae/index.php?page=source-viewer.php
Method	POST
Attack	
Evidence	$1$15726933$wR5Lg9fHFoYgxMlSeIK8Z.
URL	http://192.168.233.128/mutillidae/index.php?page=source-viewer.php
Method	POST
Attack	
Evidence	$1$18994812$m57gS66pqpLZUujk.1y6H/
URL	http://192.168.233.128/mutillidae/index.php?page=source-viewer.php
Method	POST
Attack	
Evidence	$1$20197093$e4HK2YiLPs22NMytU0pIp/
Instances	5
Solution	Ensure that hashes that are used to protect credentials or other resources are not leaked by the web server or database. There is typically no requirement for password hashes to be accessible to the web browser.
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage http://openwall.info/wiki/john/sample-hashes
CWE Id	200
WASC Id	13
Plugin Id	10097

High	Path Traversal
	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

Description	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://192.168.233.128/mutillidae/index.php?page=%2Fetc%2Fpasswd
Method	GET
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 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." For filenames, use stringent allow lists 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 an allow list 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 allow list 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
Plugin Id	6

High	**Remote Code Execution - CVE-2012-1823**
Description	Some PHP versions, when configured to run using CGI, do not correctly handle query strings that lack an unescaped "=" character, enabling arbitrary code execution. In this case, an operating system command was caused to be executed on the web server, and the results were returned to the web browser.
URL	http://192.168.233.128/mutillidae/?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input
Method	POST
Attack	<?php exec('echo eh1e9x72hlw7xrp5xgsx',$colm);echo join(" ",$colm);die();?>
Evidence	eh1e9x72hlw7xrp5xgsx
URL	http://192.168.233.128/mutillidae/index.php?-d+allow_url_include%3d1+-d+auto_prepend_file%3dphp://input
Method	POST
Attack	<?php exec('echo eh1e9x72hlw7xrp5xgsx',$colm);echo join(" ",$colm);die();?>
Evidence	eh1e9x72hlw7xrp5xgsx
Instances	2
Solution	Upgrade to the latest stable version of PHP, or use the Apache web server and the mod_rewrite module to filter out malicious requests using the "RewriteCond" and "RewriteRule" directives.
Reference	http://projects.webappsec.org/Improper-Input-Handling http://cwe.mitre.org/data/definitions/89.html
CWE Id	20
WASC Id	20
Plugin Id	20018

High	**Source Code Disclosure - CVE-2012-1823**
Description	Some PHP versions, when configured to run using CGI, do not correctly handle query strings that lack an unescaped "=" character, enabling PHP source code disclosure, and arbitrary code execution. In this case, the contents of the PHP file were served directly to the web browser. This output will typically contain PHP, although it may also contain straight HTML.
URL	http://192.168.233.128/mutillidae/?-s
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/mutillidae/index.php?-s

Method	GET
Attack	
Evidence	
Instances	2
Solution	Upgrade to the latest stable version of PHP, or use the Apache web server and the mod_rewrite module to filter out malicious requests using the "RewriteCond" and "RewriteRule" directives.
Reference	http://projects.webappsec.org/Improper-Input-Handling http://cwe.mitre.org/data/definitions/89.html
CWE Id	20
WASC Id	20
Plugin Id	20017

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://192.168.233.128/dvwa/login.php
Method	GET
Attack	
Evidence	<form action="login.php" method="post">
URL	http://192.168.233.128/mutillidae/?page=add-to-your-blog.php
Method	GET
Attack	
Evidence	<form action="index.php?page=add-to-your-blog.php" method="post" enctype="application/x-www-form-urlencoded" onsubmit="return onSubmitBlogEntry(this);" id="idBlogForm">
URL	http://192.168.233.128/mutillidae/?page=login.php
Method	GET
Attack	
Evidence	<form action="index.php?page=login.php" method="post" enctype="application/x-www-form-urlencoded" onsubmit="return onSubmitOfLoginForm(this);" id="idLoginForm">
URL	http://192.168.233.128/mutillidae/?page=register.php

Method	GET
Attack	
Evidence	<form action="index.php?page=register.php" method="post" enctype="application/x-www-form-urlencoded">
URL	http://192.168.233.128/mutillidae/?page=text-file-viewer.php
Method	GET
Attack	
Evidence	<form action="index.php?page=text-file-viewer.php" method="post" enctype="application/x-www-form-urlencoded">
URL	http://192.168.233.128/mutillidae/?page=user-info.php
Method	GET
Attack	
Evidence	<form action="./index.php?page=user-info.php" method="POST" enctype="application/x-www-form-urlencoded" >
URL	http://192.168.233.128/mutillidae/?page=view-someones-blog.php
Method	GET
Attack	
Evidence	<form action="index.php?page=view-someones-blog.php" method="post" enctype=" application/x-www-form-urlencoded">
URL	http://192.168.233.128/mutillidae/index.php?choice=nmap&initials=ZAP&page=user-poll.php&user-poll-php-submit-button=Submit+Vote
Method	GET
Attack	
Evidence	<form action="index.php" method="GET" enctype="application/x-www-form-urlencoded" id=" idPollForm">
URL	http://192.168.233.128/mutillidae/index.php?page=add-to-your-blog.php
Method	GET
Attack	
Evidence	<form action="index.php?page=add-to-your-blog.php" method="post" enctype="application /x-www-form-urlencoded" onsubmit="return onSubmitBlogEntry(this);" id="idBlogForm">
URL	http://192.168.233.128/mutillidae/index.php?page=dns-lookup.php
Method	GET
Attack	
Evidence	<form action="index.php?page=dns-lookup.php" method="post" enctype="application/x-www-form-urlencoded" onsubmit="return onSubmitBlogEntry(this);" id="idDNSLookupForm" >
URL	http://192.168.233.128/mutillidae/index.php?page=html5-storage.php
Method	GET
Attack	
Evidence	<form action="index.php?page=html5-storage.php" method="post" enctype="application/x-www-form-urlencoded" onsubmit="return false;" id="idForm">
URL	http://192.168.233.128/mutillidae/index.php?page=login.php
Method	GET
Attack	
Evidence	<form action="index.php?page=login.php" method="post" enctype="application/x-www-form-urlencoded" onsubmit="return onSubmitOfLoginForm(this);" id="idLoginForm">

Medium	Application Error Disclosure
Description	This page contains an error/warning message that may disclose sensitive information like the location of the file that produced the unhandled exception. This information can be used to launch further attacks against the web application. The alert could be a false positive if the error message is found inside a documentation page.
URL	http://192.168.233.128/dav/
Method	GET
Attack	
Evidence	Parent Directory
URL	http://192.168.233.128/dav/?C=D;O=A
Method	GET
Attack	
Evidence	Parent Directory
URL	http://192.168.233.128/dav/?C=M;O=A
Method	GET
Attack	
Evidence	Parent Directory
URL	http://192.168.233.128/dav/?C=N;O=D
Method	GET
Attack	
Evidence	Parent Directory
URL	http://192.168.233.128/dav/?C=S;O=A
Method	GET
Attack	
Evidence	Parent Directory
URL	http://192.168.233.128/mutillidae/?page=add-to-your-blog.php
Method	GET

URL	http://192.168.233.128/mutillidae/index.php?page=pen-test-tool-lookup.php
Method	GET
Attack	
Evidence	Fatal error: Call to a member function fetch_object() on a non-object in /var /www/mutillidae/pen-test-tool-lookup.php on line 273
URL	http://192.168.233.128/mutillidae/index.php?page=show-log.php
Method	GET
Attack	
Evidence	Table 'metasploit.hitlog' doesn't exist
URL	http://192.168.233.128/mutillidae/index.php?page=user-info.php&password=ZAP&user-info-php-submit-button=View+Account+Details&username=ZAP
Method	GET
Attack	
Evidence	Table 'metasploit.accounts' doesn't exist
URL	http://192.168.233.128/mutillidae/index.php?page=view-someones-blog.php
Method	GET
Attack	
Evidence	Table 'metasploit.accounts' doesn't exist
URL	http://192.168.233.128/mutillidae/index.php?page=add-to-your-blog.php
Method	POST
Attack	
Evidence	Table 'metasploit.blogs_table' doesn't exist
URL	http://192.168.233.128/mutillidae/index.php?page=login.php
Method	POST
Attack	
Evidence	Warning: Cannot modify header information - headers already sent by (output started at /var/www/mutillidae/process-login-attempt.php:97) in /var/www/mutillidae /index.php on line 148
URL	http://192.168.233.128/mutillidae/index.php?page=pen-test-tool-lookup.php

Method	POST
Attack	
Evidence	Fatal error: Call to a member function fetch_object() on a non-object in /var/www/mutillidae/pen-test-tool-lookup.php on line 273
URL	http://192.168.233.128/mutillidae/index.php?page=register.php
Method	POST
Attack	
Evidence	Table 'metasploit.accounts' doesn't exist
URL	http://192.168.233.128/mutillidae/index.php?page=source-viewer.php
Method	POST
Attack	
Evidence	Warning: highlight_file(1) [function.highlight-file]: failed to open stream: No such file or directory in /var/www/mutillidae/source-viewer.php on line 214
URL	http://192.168.233.128/mutillidae/index.php?page=text-file-viewer.php
Method	POST
Attack	
Evidence	Warning: fopen(2) [function.fopen]: failed to open stream: No such file or directory in /var/www/mutillidae/text-file-viewer.php on line 115
URL	http://192.168.233.128/mutillidae/index.php?page=user-info.php
Method	POST
Attack	
Evidence	Table 'metasploit.accounts' doesn't exist
URL	http://192.168.233.128/mutillidae/index.php?page=view-someones-blog.php
Method	POST
Attack	
Evidence	Table 'metasploit.accounts' doesn't exist
Instances	21
Solution	Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.
Reference	
CWE Id	200
WASC Id	13
Plugin Id	90022

Medium	**Content Security Policy (CSP) Header Not Set**
Description	Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.
URL	http://192.168.233.128/
Method	GET

Attack	
Evidence	
URL	http://192.168.233.128/*
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/*;q=0.8
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/--
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/b
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=D;O=A
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=M;O=A
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=N;O=D
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=S;O=A
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/div
Method	GET
Attack	

Instances	141
Solution	Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

Reference	https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html http://www.w3.org/TR/CSP/ http://w3c.github.io/webappsec/specs/content-security-policy/csp-specification.dev.html http://www.html5rocks.com/en/tutorials/security/content-security-policy/ http://caniuse.com/#feat=contentsecuritypolicy http://content-security-policy.com/
CWE Id	693
WASC Id	15
Plugin Id	10038

Medium	Directory Browsing
Description	It is possible to view a listing of the directory contents. Directory listings may reveal hidden scripts, include files, backup source files, etc., which can be accessed to reveal sensitive information.
URL	http://192.168.233.128/dav/
Method	GET
Attack	
Evidence	<title>Index of /dav</title>
URL	http://192.168.233.128/dav/?C=D;O=A
Method	GET
Attack	
Evidence	<title>Index of /dav</title>
URL	http://192.168.233.128/dav/?C=M;O=A
Method	GET
Attack	
Evidence	<title>Index of /dav</title>
URL	http://192.168.233.128/dav/?C=N;O=D
Method	GET
Attack	
Evidence	<title>Index of /dav</title>
URL	http://192.168.233.128/dav/?C=S;O=A
Method	GET
Attack	
Evidence	<title>Index of /dav</title>
Instances	5
Solution	Configure the web server to disable directory browsing.
Reference	https://cwe.mitre.org/data/definitions/548.html
CWE Id	548
WASC Id	16
Plugin Id	10033

Medium	Hidden File Found
Description	A sensitive file was identified as accessible or available. This may leak administrative, configuration, or credential information which can be leveraged by a malicious individual to further attack the system or conduct social engineering efforts.

URL	http://192.168.233.128/mutillidae/phpinfo.php
Method	GET
Attack	
Evidence	HTTP/1.1 200 OK
URL	http://192.168.233.128/phpinfo.php
Method	GET
Attack	
Evidence	HTTP/1.1 200 OK
Instances	2
Solution	Consider whether or not the component is actually required in production, if it isn't then disable it. If it is then ensure access to it requires appropriate authentication and authorization, or limit exposure to internal systems or specific source IPs, etc.
Reference	https://blog.hboeck.de/archives/892-Introducing-Snallygaster-a-Tool-to-Scan-for-Secrets-on-Web-Servers.html https://www.php.net/manual/en/function.phpinfo.php
CWE Id	538
WASC Id	13
Plugin Id	40035

Medium	**Missing Anti-clickjacking Header**
Description	The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
URL	http://192.168.233.128/
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=D;O=A
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=M;O=A
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=N;O=D
Method	GET
Attack	
Evidence	
URL	http://192.168.233.128/dav/?C=S;O=A
Method	GET

Instances	86
Solution	Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. 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
Plugin Id	10020

Medium	**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://192.168.233.128/mutillidae/index.php?page=%40
Method	GET
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
Plugin Id	40008

Medium	**Vulnerable JS Library**
Description	The identified library jquery, version 1.3.2 is vulnerable.
URL	http://192.168.233.128/mutillidae/javascript/ddsmoothmenu/jquery.min.js
Method	GET
Attack	
Evidence	* jQuery JavaScript Library v1.3.2
Instances	1
Solution	Please upgrade to the latest version of jquery.
Reference	https://nvd.nist.gov/vuln/detail/CVE-2012-6708 http://research.insecurelabs.org/jquery/test/ https://bugs.jquery.com/ticket/9521 http://bugs.jquery.com/ticket/11290 https://blog.jquery.com/2019/04/10/jquery-3-4-0-released/ https://nvd.nist.gov/vuln/detail/CVE-2019-11358 https://github.com/advisories/GHSA-q4m3-2j7h-f7xw https://github.com/jquery/jquery/commit/753d591aea698e57d6db58c9f722cd0808619b1b https://blog.jquery.com/2020/04/10/jquery-3-5-0-released/

	https://github.com/jquery/jquery.com/issues/162 https://nvd.nist.gov/vuln/detail/CVE-2020-7656 https://nvd.nist.gov/vuln/detail/CVE-2011-4969
CWE Id	829
WASC Id	
Plugin Id	10003

Medium	XSLT Injection
Description	Injection using XSL transformations may be possible, and may allow an attacker to read system information, read and write files, or execute arbitrary code.
URL	http://192.168.233.128/mutillidae/index.php?page=%3Cxsl%3Avalue-of+select%3D%22document%28%27http%3A%2F%2F192.168.233.128%3A22%27%29%22%2F%3E
Method	GET
Attack	<xsl:value-of select="document('http://192.168.233.128:22')"/>
Evidence	failed to open stream
Instances	1
Solution	Sanitize and analyze every user input coming from any client-side.
Reference	https://www.contextis.com/blog/xslt-server-side-injection-attacks
CWE Id	91
WASC Id	23
Plugin Id	90017