SQL Injection	
Risk Level	High
Description	• The parameters – **blog, entry, password**, and **username** appear to be vulnerable to SQL injection attacks. A single quote was submitted in the username parameter, and a database error message was returned. Two single quotes were then submitted and the error message disappeared.
Impact	i) Sensitive information such as the user information, OS, network configuration etc can be disclosed to the attacker. ii) The attacker can gain access to the server and can even take control of the entire system. iii) DoS attacks can be launched on the server using resource-intensive and infinite loops.
Recommendation	It is recommended to implement the following configurations: 1. **Input Validation:** Comparing against a whitelist of permitted values, allowing only alphanumeric characters in the field and not any other syntax. 2. **Least Privilege:** The users of the application should get the least privilege possible so that the command cannot cause any substantial damage. 3. **Error Messages:** It's recommended to not display SQL Error messages in the responses.
Affected URL and Ports	http://192.168.56.128/mutillidae/index.php

File Traversal (page parameter)	
Risk Level	High
Description	The page parameter is vulnerable to path traversal attacks, enabling read access to arbitrary files on the server. The payload ../../../../../../../../../../../../../../../../**etc/passwd** was submitted in the page parameter. The requested file was returned in the application's response.
Impact	i) Sensitive information such as the user information, OS, network configuration etc can be disclosed to the attacker. ii) Attacker may also gain access to files containing configuration data, passwords, database records, log data, source code, and program scripts and binaries.
Recommendation	It is recommended to implement the following configurations: 1. After validating user input, the application can use a suitable file system API to verify that the file to be accessed is actually located within the base directory used by the application. 2. User-controllable data should be strictly validated before being passed to any file system operation. In particular, input containing dot-dot sequences should be blocked. 3. The directory used to store files that are accessed using user-controllable data can be located on a separate logical volume to other sensitive application and operating system files, so that these cannot be reached via path traversal attacks. a. For Unix – a chrooted file system b. For Windows - mounting the base directory as a new logical drive and using the associated drive letter to access its contents.
Affected URL and Ports	http://192.168.56.128/mutillidae/ [page parameter] http://192.168.56.128/mutillidae/index.php [page parameter]

File Path Manipulation	
Risk Level	High
Description	The **page** parameter appears to be vulnerable to file path manipulation attacks. The payload .htaccess was submitted in the page parameter. The file .htaccess was returned.
Impact	i) Sensitive information such as the user information, OS, network configuration etc can be disclosed to the attacker. ii) Attacker may be able to retrieve items that are normally protected from direct access, such as application configuration files, the source code for server-executable scripts, or files with extensions that the web server is not configured to serve directly.
Recommendation	• Ideally, application functionality should be designed in such a way that user-controllable data does not need to be placed into file or URL paths in order to access local resources on the server. This can normally be achieved by referencing known files via an index number rather than their name. • If it is considered unavoidable to place user data into file or URL paths, the data should be strictly validated against a whitelist of accepted values.
Affected URL and Ports	http://192.168.56.128/mutillidae/ [page parameter] http://192.168.56.128/mutillidae/index.php [page parameter]

Client-Side Desync	
Risk Level	High
Description	The server appears to be vulnerable to client-side desync attacks. A POST request was sent to the path '/mutillidae/documentation/mutillidae-installation-on-xampp-win7.pdf' with a second request sent as the body. The server ignored the Content-Length header and did not close the connection, leading to the smuggled request being interpreted as the next request.
Impact	• The web server fails to correctly process the Content-Length of POST requests. Exploiting this behavior, an attacker can force a victim's browser to desynchronize its connection with the website, typically leading to XSS.
Recommendation	• You can resolve this vulnerability by patching the server so that it either processes POST requests correctly, or closes the connection after handling them. • You could also disable connection reuse entirely, but this may reduce performance. You can also resolve this issue by enabling HTTP/2.
Affected URL and Ports	http://192.168.56.128/mutillidae/documentation/mutillidae-installation-on-xampp-win7.pdf

Cross-Site Scripting (reflected)	
Risk Level	High
Description	• Reflected cross-site scripting vulnerabilities occur when data is echoed into an application's response without proper validation. • Attackers can exploit this vulnerability to execute their JavaScript code within a user's browser session. • The attacker's code can perform malicious actions, including stealing session tokens, login credentials, and logging keystrokes. • Users can be tricked into executing the attacker's request through various means, such as malicious URLs in emails or instant messages. • The impact of cross-site scripting vulnerabilities depends on the application's nature, the data it contains, and its association with other applications. It can range from low risk for non-sensitive content-only applications to high risk for applications with access to critical functionality or belonging to targeted organizations.
Impact	i) Sensitive information such as the user information, OS, network configuration etc can be disclosed to the attacker. ii) Attacker may be able to retrieve items that are normally protected from direct access, such as application configuration files, the source code for server-executable scripts, or files with extensions that the web server is not configured to serve directly.
Recommendation	• Input should be validated as strictly as possible on arrival, given the kind of content that it is expected to contain. For example, personal names should consist of alphabetical and a small range of typographical characters, and be relatively short; a year of birth should consist of exactly four numerals; email addresses should match a well-defined regular expression. Input which fails the validation should be rejected, not sanitized. • User input should be HTML-encoded at any point where it is copied into application responses. All HTML metacharacters, including < > " ' and =, should be replaced with the corresponding HTML entities (&lt; &gt; etc).
Affected URL and Ports	1. **http://192.168.56.128/mutillidae/** [page parameter] 2. **http://192.168.56.128/mutillidae/index.php** [blog_entry parameter] 3. **http://192.168.56.128/mutillidae/index.php** [page parameter] 4. **http://192.168.56.128/mutillidae/index.php** [page parameter] 5. **http://192.168.56.128/mutillidae/index.php** [page parameter] 6. **http://192.168.56.128/mutillidae/index.php** [password parameter] 7. **http://192.168.56.128/mutillidae/index.php** [username parameter] 8. **http://192.168.56.128/mutillidae**/ [Referer HTTP header] 9. **http://192.168.56.128/mutillidae/** [User-Agent HTTP header] 10. **http://192.168.56.128/mutillidae/index.php** [Referer HTTP header] 11. **http://192.168.56.128/mutillidae/index.php** [Referer HTTP header] 12. **http://192.168.56.128/mutillidae/index.php** [User-Agent HTTP header]

Cross-Site Scripting (DOM-Based)	
Risk Level	High
Description	• The application may be vulnerable to DOM-based cross-site scripting. Data is read from **input.value** and passed to **element.innerHTML**.
Impact	• The attacker-supplied code can perform a wide variety of actions, such as stealing the victim's session token or login credentials, performing arbitrary actions on the victim's behalf, and logging their keystrokes.
Recommendation	• To avoid DOM-based cross-site scripting vulnerabilities, refrain from dynamically writing untrusted data into the HTML document. • If unavoidable, implement client-side defenses to prevent malicious data from introducing script code. • Validate data on a whitelist basis to allow only known safe content. • Sanitize or encode the data, which may involve JavaScript escaping, HTML encoding, and URL encoding. • Context matters when determining the appropriate sequence for encoding or escaping.
Affected URL and Ports	**http://192.168.56.128/mutillidae/index.php**

Clear text Password Submission	
Risk Level	High
Description	• The page contains a form with the following action URL, which is submitted over clear-text HTTP: http://192.168.56.128/mutillidae/index.php?page=login.php • The form contains the following password field: password • This issue was found in multiple locations under the reported path.
Impact	• Applications transmitting passwords over unencrypted connections are vulnerable to interception. • Attackers can exploit this vulnerability by eavesdropping on the victim's network traffic. • Insecure connections like public Wi-Fi or compromised shared networks facilitate such attacks. • Common defences like switched networks are insufficient to prevent interception. • Advanced adversaries positioned at the user's ISP or application's hosting infrastructure can also exploit this vulnerability. • Disclosing passwords can lead to compromised accounts and challenging investigations. • Reused passwords put users at risk, even if the application handles non-sensitive information.
Recommendation	• Applications should utilize transport-level encryption (SSL/TLS) to safeguard sensitive communications. • Encryption should be applied to login mechanisms, privileged actions, and functions accessing sensitive data. • Implement a secure session handling mechanism for these areas. • Avoid transmitting session tokens over unencrypted communications. • If using HTTP cookies for session tokens, set the secure flag to prevent transmission over clear-text HTTP.
Affected URL and Ports	http://192.168.56.128/mutillidae

Password submitted using GET method	
Risk Level	Low
Description	The page contains a form with the following action URL, which is submitted using the GET method: • http://192.168.56.128/mutillidae/index.php?page=user-info.php The form contains the following password field: • password This issue was found in multiple locations under the reported path.
Impact	• Passwords transmitted via GET requests are visible in various places, including browser history, server logs, and network traffic. This increases the risk of unauthorized access to sensitive information. • Attackers can exploit this vulnerability by intercepting or sniffing network traffic to capture passwords, potentially leading to unauthorized account access. • Insecure storage of GET request logs or mishandling of URL parameters can expose passwords to unauthorized individuals. • If users unknowingly submit passwords via GET requests, they may assume their information is secure, leading to a false sense of security. • Compliance with security standards and regulations, such as PCI DSS for handling payment card information, may be compromised due to the vulnerability.
Recommendation	• Always transmitting passwords via the more secure POST method instead of GET. • Educating developers and users about the risks associated with submitting passwords via GET requests. • Implementing secure transmission protocols like SSL/TLS to encrypt communication and protect sensitive data. • Regularly reviewing and securing server logs and monitoring for unauthorized access or data breaches. • Enforcing best practices for password handling and storage, such as using strong encryption and hashing algorithms.
Affected URL and Ports	http://192.168.56.128/mutillidae

Cookie without HttpOnly Flag set	
Risk Level	Low
Description	The following cookie was issued by the application and does not have the HttpOnly flag set: • **PHPSESSID** The cookie appears to contain a session token, which may increase the risk associated with this issue. You should review the contents of the cookie to determine its function.
Impact	If the HttpOnly attribute is set on a cookie, then the cookie's value cannot be read or set by client-side JavaScript. This measure makes certain client-side attacks, such as cross-site scripting, slightly harder to exploit by preventing them from trivially capturing the cookie's value via an injected script.
Recommendation	• There is usually no good reason not to set the HttpOnly flag on all cookies. Unless user specifically requires legitimate client-side scripts within their application to read or set a cookie's value, user should set the HttpOnly flag by including this attribute within the relevant Set-cookie directive. • User should be aware that the restrictions imposed by the HttpOnly flag can potentially be circumvented in some circumstances, and that numerous other serious attacks can be delivered by client-side script injection, aside from simple cookie stealing.
Affected URL and Ports	http://192.168.56.128/mutillidae/ [page parameter] http://192.168.56.128/mutillidae/index.php [page parameter]

Password field with autocomplete enabled	
Risk Level	Low
Description	The page contains a form with the following action URL: • http://192.168.56.128/mutillidae/index.php?page=login.php The form contains the following password field with autocomplete enabled: • password This issue was found in multiple locations under the reported path.
Impact	• Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application. • The stored credentials can be captured by an attacker who gains control over the user's computer. Further, an attacker who finds a separate application vulnerability such as cross-site scripting may be able to exploit this to retrieve a user's browser-stored credentials.
Recommendation	• To prevent browsers from storing credentials entered into HTML forms, include the attribute **autocomplete="off"** within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields). • Please note that modern web browsers may ignore this directive. In spite of this there is a chance that not disabling autocomplete may cause problems obtaining PCI compliance.
Affected URL and Ports	http://192.168.56.128/mutillidae/

Client-side HTTP parameter pollution (reflected)	
Risk Level	Low
Description	• The value of the **page** request parameter is copied into the response within the query string of a URL. ○ The payload **hjq&hue=1** was submitted in the page parameter. This input was echoed unmodified within the "href" attribute of an "a" tag. ○ This proof-of-concept attack demonstrates that it is possible to inject arbitrary query string parameters into URLs in the application's response. • The value of the **page** request parameter is copied into the response within the query string of a URL. ○ The payload **nbs&lhs=1** was submitted in the page parameter. This input was echoed unmodified within the "href" attribute of an "a" tag. ○ This proof-of-concept attack demonstrates that it is possible to inject arbitrary query string parameters into URLs in the application's response.
Impact	The security impact of this issue depends largely on the nature of the application functionality. Even if it has no direct impact on its own, an attacker may use it in conjunction with other vulnerabilities to escalate their overall severity.
Recommendation	• Ensure that user input is URL-encoded before it is embedded in a URL.
Affected URL and Ports	http://192.168.56.128/mutillidae/ [page parameter] http://192.168.56.128/mutillidae/index.php [page parameter]

Content type incorrectly stated	
Risk Level	Low
Description	The response states that the content type is **application/octet-stream**. However, it actually appears to contain **unrecognized content**. The following browsers may interpret the response as HTML: • Internet Explorer 11 • Internet Explorer 11 (Compatibility Mode) • Edge This issue was found in multiple locations under the reported path.
Impact	If a response specifies an incorrect content type then browsers may process the response in unexpected ways. If the content type is specified to be a renderable text-based format, then the browser will usually attempt to interpret the response as being in that format, regardless of the actual contents of the response.
Recommendation	• For every response containing a message body, the application should include a single Content-type header that correctly and unambiguously states the MIME type of the content in the response body. • Additionally, the response header "X-content-type-options: nosniff" should be returned in all responses to reduce the likelihood that browsers will interpret content in a way that disregards the Content-type header.
Affected URL and Ports	/ /analytics-track-digest256/114.0/1683907587 /base-cryptomining-track-digest256/114.0/1683907587 /base-email-track-digest256/114.0/1683907587 /base-fingerprinting-track-digest256/114.0/1683907587 /content-email-track-digest256/114.0/1683907587 /content-track-digest256/114.0/1683907587 /google-trackwhite-digest256/114.0/1683907587 /social-track-digest256/114.0/1683907587 /social-tracking-protection-facebook-digest256/114.0/1683907587

Unencrypted Communications	
Risk Level	Low
Description	The application allows users to connect to it over unencrypted connections.
Impact	• An attacker suitably positioned to view a legitimate user's network traffic could record and monitor their interactions with the application and obtain any information the user supplies. • An attacker able to modify traffic could use the application as a platform for attacks against its users and third-party websites. • Using a mixture of encrypted and unencrypted communications is an ineffective defense against active attackers, because they can easily remove references to encrypted resources when these references are transmitted over an unencrypted connection.
Recommendation	• Applications should use transport-level encryption (SSL/TLS) to protect all communications passing between the client and the server. • The Strict-Transport-Security HTTP header should be used to ensure that clients refuse to access the server over an insecure connection.
Affected URL and Ports	http://127.0.0.1:9093/ http://192.168.56.128/

Strict transport security not enforced	
Risk Level	Low
Description	The application fails to prevent users from connecting to it over unencrypted connections. An attacker able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption, and use the application as a platform for attacks against its users.
Impact	• An attacker able to modify a legitimate user's network traffic could bypass the application's use of SSL/TLS encryption, and use the application as a platform for attacks against its users.
Recommendation	• The application should instruct web browsers to only access the application using HTTPS. ○ Enable HTTP Strict Transport Security (HSTS) by adding a response header with the name 'Strict-Transport-Security' and the value 'max-age=expireTime', where expireTime is the time in seconds that browsers should remember that the site should only be accessed using HTTPS. Consider adding the 'includeSubDomains' flag if appropriate
Affected URL and Ports	• https://content-signature-2.cdn.mozilla.net/chains/normandy.content-signature.mozilla.org-2023-08-09-20-34-35.chain • https://getpocket.cdn.mozilla.net/v3/firefox/global-recs • https://push.services.mozilla.com/ • https://safebrowsing.googleapis.com/v4/threatListUpdates:fetch • https://tracking-protection.cdn.mozilla.net/ • https://tracking-protection.cdn.mozilla.net/analytics-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/base-cryptomining-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/base-email-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/base-fingerprinting-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/content-email-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/content-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/google-trackwhite-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/social-track-digest256/114.0/1683907587 • https://tracking-protection.cdn.mozilla.net/social-tracking-protection-facebook-digest256/114.0/1683907587

Path-relative style sheet import	
Risk Level	Information
Description	The **page** parameter appears to be vulnerable to file path manipulation attacks. The payload .htaccess was submitted in the page parameter. The file .htaccess was returned.
Impact	The application may be vulnerable to path-relative style sheet import (PRSSI) attacks. Being able to inject arbitrary CSS into the victim's browser may enable various attacks, including: • Executing arbitrary JavaScript using IE's expression() function. • Using CSS selectors to read parts of the HTML source, which may include sensitive data such as anti-CSRF tokens. • Capturing any sensitive data within the URL query string by making a further style sheet import to a URL on the attacker's domain, and monitoring the incoming Referer header.
Recommendation	The root cause of the vulnerability can be resolved by not using path-relative URLs in style sheet imports. Aside from this, attacks can also be prevented by implementing all of the following defensive measures: • Setting the HTTP response header "X-Frame-Options: deny" in all responses. One method that an attacker can use to make a page render in quirks mode is to frame it within their own page that is rendered in quirks mode. Setting this header prevents the page from being framed. • Setting a modern doctype (e.g. "<!doctype html>") in all HTML responses. This prevents the page from being rendered in quirks mode (unless it is being framed, as described above). • Setting the HTTP response header "X-Content-Type-Options: nosniff" in all responses. This prevents the browser from processing a non-CSS response as CSS, even if another page loads the response via a style sheet import.
Affected URL and Ports	http://192.168.56.128/mutillidae/ http://192.168.56.128/mutillidae/index.php http://192.168.56.128/mutillidae/framer.html

External service interaction (DNS)	
Risk Level	Information
Description	• It is possible to induce the application to perform server-side DNS lookups of arbitrary domain names. • The payload 0efnu2pjgj2kt8iijmz0oe7ck3qyeo2iw6mtch1.oastify.com was submitted in the HTTP Host header. • The application performed a DNS lookup of the specified domain.
Impact	i) Sensitive information such as the user information, OS, network configuration etc can be disclosed to the attacker. ii) Attacker may be able to retrieve items that are normally protected from direct access, such as application configuration files, the source code for server-executable scripts, or files with extensions that the web server is not configured to serve directly.
Recommendation	• Review the purpose of application functionality and assess if arbitrary external service interactions are intended. Understand potential attack types and take appropriate measures, such as blocking network access to internal systems and securing the application server. If unintended, implement a whitelist of allowed services and hosts, blocking unauthorized interactions. • Out-of-Band Application Security Testing (OAST) is effective for finding high-risk features, but locating the root cause can be challenging. Determine if interactions are triggered by specific functionality or occur universally. CDN, application firewall, analytics systems, or third-party systems may contribute to interactions. For instance, an HTTP request might trigger a malicious email passing through a link-scanner before reaching the recipient.
Affected URL and Ports	http://192.168.56.128/mutillidae/

Referer-dependent response	
Risk Level	Information
Description	• Application responses may depend systematically on the presence or absence of the Referer header in requests. This behavior does not necessarily constitute a security vulnerability, and you should investigate the nature of and reason for the differential responses to determine whether a vulnerability is present.
Impact	Possible reasons for Referer-dependent responses are: • Referer-based access controls assume that arriving from a privileged location grants authorization to access another privileged area. However, this can be easily bypassed by including an accepted Referer header in requests. • Protection against cross-site request forgery attacks involves verifying that privileged actions originate from within the application. Removing the Referer header completely can often bypass these defenses. • Referer-tailored content, such as domain-specific welcome messages and SEO techniques, aims to personalize the user experience. However, mishandling the Referer header can lead to vulnerabilities like SQL injection and cross-site scripting. Manipulating search terms in the Referer header can result in persistent code injection attacks.
Recommendation	• The Referer header is not a robust foundation on which to build access controls. Any such measures should be replaced with more secure alternatives that are not vulnerable to Referer spoofing. • If the contents of responses is updated based on Referer data, then the same defenses against malicious input should be employed here as for any other kinds of user-supplied data.
Affected URL and Ports	http://192.168.56.128/mutillidae/ http://192.168.56.128/mutillidae/index.php

User agent-dependent response	
Risk Level	Information
Description	• Application responses may depend systematically on the value of the User-Agent header in requests.
Impact	• This behavior does not itself constitute a security vulnerability, but may point towards additional attack surface within the application, which may contain vulnerabilities. • This behavior often arises because applications provide different user interfaces for desktop and mobile users. Mobile interfaces have often been less thoroughly tested for vulnerabilities such as cross-site scripting, and often have simpler authentication and session handling mechanisms that may contain problems that are not present in the full interface.
Recommendation	• Diversify authentication: Avoid relying solely on User-Agent for user access. Implement secure authentication methods like tokens, session management, or username/password combinations. • Validate input: Perform thorough input validation and sanitization, including the User-Agent header, to prevent injection attacks and ensure malicious input doesn't compromise the application. • Stay updated and test: Regularly update and patch the application and server, and perform security testing, including penetration testing, to identify and address vulnerabilities related to User-Agent handling.
Affected URL and Ports	http://192.168.56.128/mutillidae/ http://192.168.56.128/mutillidae/index.php

Input returned in response (Reflected)	
Risk Level	Information
Description	Reflection of input arises when data is copied from a request and echoed into the application's immediate response. Few examples: • The value of the **Referer** HTTP header is copied into the application's response. • The value of the URL path folder 1 is copied into the application's response.
Impact	i) Sensitive information such as the user information, OS, network configuration etc can be disclosed to the attacker. ii) Attacker may be able to retrieve items that are normally protected from direct access, such as application configuration files, the source code for server-executable scripts, or files with extensions that the web server is not configured to serve directly.
Recommendation	• Input sanitization: Implement strict input validation and filtering to ensure that any user-supplied input is properly sanitized, removing or encoding any potentially harmful characters or scripts. • Output encoding: Apply proper output encoding to any user-generated content displayed in the response to prevent the execution of malicious scripts or unintended HTML rendering. • Use parameterized queries: Employ parameterized queries or prepared statements when interacting with databases to prevent SQL injection attacks by automatically handling user input as data rather than executable code.
Affected URL and Ports	http://192.168.56.128/mutillidae/ [Referer HTTP header] http://192.168.56.128/mutillidae/ [URL path folder 1] http://192.168.56.128/mutillidae/ [User-Agent HTTP header] http://192.168.56.128/mutillidae/ [page parameter] http://192.168.56.128/mutillidae/documentation/mutillidae-installation-on-xampp-win7.pdf [URL path filename] http://192.168.56.128/mutillidae/documentation/mutillidae-installation-on-xampp-win7.pdf [URL path folder 1] http://192.168.56.128/mutillidae/documentation/mutillidae-installation-on-xampp-win7.pdf [URL path folder 2] http://192.168.56.128/mutillidae/framer.html [URL path filename] http://192.168.56.128/mutillidae/index.php [PHPSESSID cookie] http://192.168.56.128/mutillidae/index.php [Referer HTTP header] http://192.168.56.128/mutillidae/index.php [User-Agent HTTP header] http://192.168.56.128/mutillidae/index.php [blog_entry parameter] http://192.168.56.128/mutillidae/index.php [page parameter] http://192.168.56.128/mutillidae/index.php [password parameter] http://192.168.56.128/mutillidae/index.php [username parameter]

Cross-Domain Referer Leakage	
Risk Level	Information
Description	• A cross-domain Referer leakage vulnerability occurs when a web browser includes the "Referer" header in a request for a resource from a different domain. This header reveals the URL of the resource that initiated the request, even if it contains sensitive information like session tokens. The vulnerability arises when the receiving domain is not fully trusted, potentially leading to a security compromise..
Impact	Potential disclosure of sensitive information from the originating URL to an untrusted domain. If the Referer header contains sensitive data, such as session tokens, an attacker who controls the receiving domain could capture and exploit that information. This could result in unauthorized access, session hijacking, or other security breaches, depending on the sensitivity of the leaked data.
Recommendation	• Applications should never transmit any sensitive information within the URL query string. • In addition to being leaked in the Referer header, such information may be logged in various locations and may be visible on-screen to untrusted parties. • If placing sensitive information in the URL is unavoidable, consider using the Referer-Policy HTTP header to reduce the chance of it being disclosed to third parties.
Affected URL and Ports	/mutillidae (11 instances) /mutillidae/index.php (2 instances)