

Scan Report

vdharmava/Venu-SAST-Grok-Python1

Grouped By: Vulnerability Type
Scanned Branch Name: main
Project Created: 24 Jun, 2025 | 3:55 AM UTC+0
Last Scanned: 24 Jun, 2025 | 3:55 AM UTC+0
Scanners: SAST, IaC, Containers, SCA, SCS

C

8

H

9

M

5

22

Table of Contents

Filtered By3

Scan Information3

Project & Scan Tags3

Scan Results Overview4

By Scanner4

By Status4

By Severity4

By State4

By Language4

By Technology4

By Package4

By SAST Vulnerability4

Top 10 SAST Vulnerabilities5

Top 10 SAST Vulnerable Files5

5 Oldest SAST Vulnerabilities5

SAST Vulnerabilities7

By Severity7

SAST Scan Results7

Stored_XSS7

SQL_Injection9

Command_Injection12

Reflected_XSS13

SSRF15

Path_Traversal16

Use_Of_Hardcoded_Password17

Missing_HSTS_Header18

IaC Vulnerabilities19

By Severity19

IaC Scan Results19

SCA Vulnerabilities20

By Severity20

SCA Scan Results20

SAST Resolved Vulnerabilities21

Categories22

ASA Premium22

ASD STIG 6.122

Base Preset22

CWE top 2522

FISMA 201422

MOIS(KISA) Secure Coding 202123




NIST SP 800-5323

OWASP ASVS23

OWASP Top 10 201323

OWASP Top 10 2017	24
OWASP Top 10 2021	24
OWASP Top 10 API	24
OWASP Top 10 API 2023	24
PCI DSS v3.2.1	24
PCI DSS v4.0	25
SANS top 25	25
Top Tier	25
Vulnerability Details	26
Stored_XSS	26
SQL_Injection	26
Command_Injection	27
Reflected_XSS	27
SSRF	28
Path_Traversal	28
Use_Of_Hardcoded_Password	29
Missing_HSTS_Header	29
Supply Chain Security Vulnerabilities	31

Filtered By

Severity:   

Excluded: Low, Information

Result State: To Verify, Confirmed, Urgent

Excluded: Not Exploitable, Proposed Not Exploitable

Status: New, Recurrent

Excluded: None

Scanners: SAST, IaC, Containers, SCA, SCS

Excluded: None

Queries: [Link](#)

Results limited to: 10000

Scan Information

<ul style="list-style-type: none">▪ Scan Id: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d▪ Scan Duration: 0h 2m 7s▪ Preset: ASA High▪ LOC Scanned: 214<ul style="list-style-type: none">▪ SAST: 214▪ IaC: 0▪ Files Scanned: 11<ul style="list-style-type: none">▪ SAST: 11▪ IaC: 0▪ Density: 74.77<ul style="list-style-type: none">▪ SAST: 74.77▪ IaC: 0▪ Initiator: venu_dharmavaram▪ Online Results: Link	<ul style="list-style-type: none">▪ Source Origin: github▪ Main Branch: N/A▪ Scan Type: Full Scan▪ Scanned Branch Name: main▪ Groups: None▪ Scanner Status:<ul style="list-style-type: none">▪ IaC: Completed▪ SCS: Completed▪ SAST: Completed▪ SCA: Completed▪ Containers: Completed
--	--

Project & Scan Tags

Project Tags: None
Scan Tags: None

Scan Results Overview

By Scanner

Total
22

SAST (16, 72.73%)

IaC (0, 0%)

SCA (0, 0%)

Containers (0, 0%)

SCS (6, 27.27%)

By Status (SAST & IaC & SCA & SCS)

Total
22

New (22, 100%)

Recurrent (0, 0%)

By Severity

Total
22

Legend

Critical (8, 36.36%)

High (9, 40.91%)

Medium (5, 22.73%)

	Density (SAST & IaC)
Critical (8, 36.36%)	37.38
High (9, 40.91%)	42.06
Medium (5, 22.73%)	23.36

By State

Total
22

Legend

To Verify (22, 100.00%)

Confirmed (0, 0.00%)

Urgent (0, 0.00%)

	Density (SAST & IaC)
To Verify (22, 100.00%)	74.77
Confirmed (0, 0.00%)	0.00
Urgent (0, 0.00%)	0.00

By Language (SAST)

python (16)

8

5

3

Density (SAST)

74.77









By Technology (IaC)

No data to show

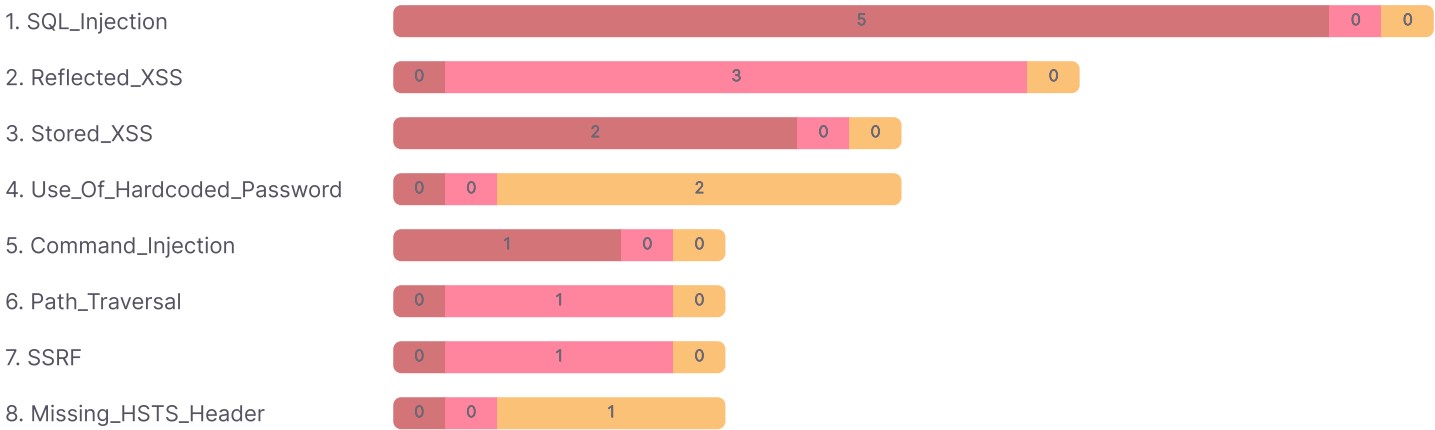
By Package (SCA)

No data to show

By SAST Vulnerability			
	Vulnerability Type		

	SQL_Injection In 1 Files	5	0	0
	Stored_XSS In 1 Files	2	0	0
	Command_Injection In 1 Files	1	0	0
	Reflected_XSS In 1 Files	0	3	0
	Path_Traversal In 1 Files	0	1	0
	SSRF In 1 Files	0	1	0
	Use_Of_Hardcoded_Password In 1 Files	0	0	2
	Missing_HSTS_Header In 1 Files	0	0	1
	Total In 2 Files	8	5	3

Top 10 SAST Vulnerabilities (16/2 Vulnerable files)



Top 10 SAST Vulnerable Files (2/11 Files)



5 Oldest SAST Vulnerabilities by severity



- 1. Command_Injection 0 days
- 2. SQL_Injection 0 days

3. Stored_XSS 0 days

1. Path_Traversal 0 days

2. Reflected_XSS 0 days

3. SSRF 0 days

1. Missing_HSTS_Header 0 days

2. Use_Of_Hardcoded_Password 0 days

SAST Vulnerabilities

By Severity



SAST Scan Results (16)

Stored_XSS (Type)

Query Path: Python/Python_Critical_Risk/Stored_XSS

CWE Id: 79

Total results: 2

Description: The method @DestinationMethod embeds untrusted data in generated output with @DestinationElement, at line @DestinationLine of @DestinationFile. This untrusted data is embedded into the output without proper sanitization or encoding, enabling an attacker to inject malicious code into the generated web-page. The attacker would be able to alter the returned web page by saving malicious data in a data-store ahead of time. The attacker's modified data is then read from the database by the @SourceMethod method with @SourceElement, at line @SourceLine of @SourceFile. This untrusted data then flows through the code straight to the output web page, without sanitization. This can enable a Stored Cross-Site Scripting (XSS) attack.

Category:

- ASA Premium: ASA Premium
- ASD STIG 6.1: APSC-DV-002490 - CAT I The application must protect from Cross-Site Scripting (XSS) vulnerabilities.
- Base Preset: Base Preset
- CWE top 25: CWE top 25
- FISMA 2014: System And Information Integrity
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
- NIST SP 800-53: SI-15 Information Output Filtering (PO)
- OWASP ASVS: V05 Validation, Sanitization and Encoding
- OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)
- OWASP Top 10 2017: A7-Cross-Site Scripting (XSS)
- OWASP Top 10 2021: A3-Injection
- PCI DSS v3.2.1: PCI DSS (3.2.1) - 6.5.7 - Cross-site scripting (XSS)
- PCI DSS v4.0: PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development
- SANS top 25: SANS top 25
- Top Tier: Top Tier

Result 1 of 2

Critical • Link • New • To Verify • Similarity Id: -105436583 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

<p>Source</p> <p>File Name: /auth_app/views.py</p> <p>Method: fetch_url</p> <p>Element: get</p>	<p>Destination</p> <p>File Name: /auth_app/views.py</p> <p>Method: fetch_url</p> <p>Element: HttpResponse</p>
<p>Code Snippets</p> <div><p>57 response = requests.get(url) # No URL validation</p><p>58 return HttpResponse(response.text)</p></div>	

Result 2 of 2

Critical • Link • New • To Verify • Similarity Id: -1038874957 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

<div>Source</div> <div>File Name: /auth_app/views.py</div> <div>Method: vulnerable_endpoint</div> <div>Element: read</div>	<div>Destination</div> <div>File Name: /auth_app/views.py</div> <div>Method: vulnerable_endpoint</div> <div>Element: HttpResponse</div>
<div>Code Snippets</div> <div><div>84 response += f.read()</div><div>100 No source code available</div></div>	

SQL_Injection (Type)

Query Path: Python/Python_Critical_Risk/SQL_Injection

CWE Id: 89

Total results: 5

Description: The application's @DestinationMethod method executes an SQL query with @DestinationElement, at line @DestinationLine of @DestinationFile. The application constructs this SQL query by embedding an untrusted string into the query without proper sanitization. The concatenated string is submitted to the database, where it is parsed and executed accordingly. An attacker would be able to inject arbitrary syntax and data into the SQL query, by crafting a malicious payload and providing it via the input @SourceElement; this input is then read by the @SourceMethod method at line @SourceLine of @SourceFile. This input then flows through the code, into a query and to the database server - without sanitization. This may enable an SQL Injection attack.

Category:

- ASA Premium: ASA Premium
- ASD STIG 6.1: APSC-DV-002540 - CAT I The application must not be vulnerable to SQL Injection.
- Base Preset: Base Preset
- CWE top 25: CWE top 25
- FISMA 2014: System And Information Integrity
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
- NIST SP 800-53: SI-10 Information Input Validation (P1)
- OWASP ASVS: V05 Validation, Sanitization and Encoding
- OWASP Top 10 2013: A1-Injection
- OWASP Top 10 2021: A3-Injection
- PCI DSS v3.2.1: PCI DSS (3.2.1) - 6.5.1 - Injection flaws - particularly SQL injection
- PCI DSS v4.0: PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development
- SANS top 25: SANS top 25
- Top Tier: Top Tier

Result 1 of 5

Critical • Link • New • To Verify • Similarity Id: -980163751 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /auth_app/views.py	File Name: /auth_app/views.py
Method: register	Method: register
Element: POST	Element: execute

Code Snippets

14 | password = request.POST.get('password')

18 | cursor.execute(query)

Result 2 of 5

Critical • Link • New • To Verify • Similarity Id: 621549221 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source

File Name: /auth_app/views.py

Method: login

Element: POST

Destination

File Name: /auth_app/views.py

Method: login

Element: execute

Code Snippets

```
26 | username = request.POST.get('username')
```

```
30 | cursor.execute(query)
```

Result 3 of 5

Critical • Link • New • To Verify • Similarity Id: 823753535 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source

File Name: /auth_app/views.py

Method: register

Element: POST

Destination

File Name: /auth_app/views.py

Method: register

Element: execute

Code Snippets

```
15 | email = request.POST.get('email')
```

```
18 | cursor.execute(query)
```

Result 4 of 5

Critical • Link • New • To Verify • Similarity Id: -1413873351 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source

File Name: /auth_app/views.py

Method: register

Element: POST

Destination

File Name: /auth_app/views.py

Method: register

Element: execute

Code Snippets

```
13 | username = request.POST.get('username')
```

```
18 | cursor.execute(query)
```

Result 5 of 5

Critical • [Link](#) • New • To Verify • Similarity Id: -1744695099 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source File Name: /auth_app/views.py Method: login Element: POST	Destination File Name: /auth_app/views.py Method: login Element: execute
Code Snippets <div>27 password = request.POST.get('password')</div> <div>30 cursor.execute(query)</div>	

Command_Injection (Type)

Query Path: Python/Python_Critical_Risk/Command_Injection

CWE Id: 77

Total results: 1

Description: The application's @DestinationMethod method calls an OS (shell) command with @DestinationElement, at line @DestinationLine of @DestinationFile, using an untrusted string with the command to execute. This could allow an attacker to inject an arbitrary command, and enable a Command Injection attack. The attacker may be able to inject the executed command via user input, @SourceElement, which is retrieved by the application in the @SourceMethod method, at line @SourceLine of @SourceFile.

Category:

- ASA Premium: ASA Premium
- ASD STIG 6.1: APSC-DV-002510 - CAT I The application must protect from command injection.
- Base Preset: Base Preset
- CWE top 25: CWE top 25
- FISMA 2014: System And Information Integrity
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
- NIST SP 800-53: SI-10 Information Input Validation (P1)
- OWASP Top 10 2013: A1-Injection
- OWASP Top 10 2021: A3-Injection
- PCI DSS v3.2.1: PCI DSS (3.2.1) - 6.5.1 - Injection flaws - particularly SQL injection
- PCI DSS v4.0: PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development
- Top Tier: Top Tier

Result 1 of 1

Critical • Link • New • To Verify • Similarity Id: -871652114 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /auth_app/views.py	File Name: /auth_app/views.py
Method: dangerous_function	Method: dangerous_function
Element: GET	Element: BinaryExpr

Code Snippets

70 | path = request.GET.get('path')

71 | os.system(f"cat {path}") # Dangerous function

Reflected_XSS (Type)

Query Path: Python/Python_High_Risk/Reflected_XSS

CWE Id: 79

Total results: 3

Description: The method @DestinationMethod embeds untrusted data in generated output with @DestinationElement, at line @DestinationLine of @DestinationFile. This untrusted data is embedded into the output without proper sanitization or encoding, enabling an attacker to inject malicious code into the generated web-page. The attacker would be able to alter the returned web page by simply providing modified data in the user input @SourceElement, which is read by the @SourceMethod method at line @SourceLine of @SourceFile. This input then flows through the code straight to the output web page, without sanitization. This can enable a Reflected Cross-Site Scripting (XSS) attack.

Category:

- ASA Premium: ASA Premium
- ASD STIG 6.1: APSC-DV-002490 - CAT I The application must protect from Cross-Site Scripting (XSS) vulnerabilities.
- CWE top 25: CWE top 25
- FISMA 2014: System And Information Integrity
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
- NIST SP 800-53: SI-15 Information Output Filtering (PO)
- OWASP ASVS: V05 Validation, Sanitization and Encoding
- OWASP Top 10 2013: A3-Cross-Site Scripting (XSS)
- OWASP Top 10 2017: A7-Cross-Site Scripting (XSS)
- OWASP Top 10 2021: A3-Injection
- PCI DSS v3.2.1: PCI DSS (3.2.1) - 6.5.7 - Cross-site scripting (XSS)
- PCI DSS v4.0: PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development
- SANS top 25: SANS top 25
- Top Tier: Top Tier

Result 1 of 3

High • Link • New • To Verify • Similarity Id: 831184788 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /auth_app/views.py	File Name: /auth_app/views.py
Method: vulnerable_endpoint	Method: vulnerable_endpoint
Element: request	Element: HttpResponse

Code Snippets

75

def vulnerable_endpoint(request):

100

No source code available

Result 2 of 3

High • Link • New • To Verify • Similarity Id: -750769019 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source File Name: /auth_app/views.py Method: parse_xml Element: request	Destination File Name: /auth_app/views.py Method: parse_xml Element: HttpResponse
Code Snippets <div>47 def parse_xml(request):</div> <div>51 return HttpResponse(f"Parsed: {tree.tag}")</div>	

Result 3 of 3

High • Link • New • To Verify • Similarity Id: 780725330 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source File Name: /auth_app/views.py Method: register Element: request	Destination File Name: /auth_app/views.py Method: register Element: HttpResponse
Code Snippets <div>10 def register(request):</div> <div>20 return HttpResponse(f"Welcome {username}")</div>	

SSRF (Type)

Query Path: Python/Python_High_Risk/SSRF

CWE Id: 918

Total results: 1

Description: The application sends a request to a remote server, for some resource, using @DestinationElement in @DestinationFile:@DestinationLine. However, an attacker can control the target of the request, by sending a URL or other data in @SourceElement at @SourceFile:@SourceLine.

Category:

- ASA Premium: ASA Premium
- CWE top 25: CWE top 25
- FISMA 2014: System And Information Integrity
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
- NIST SP 800-53: SI-10 Information Input Validation (P1)
- OWASP ASVS: V05 Validation, Sanitization and Encoding
- OWASP Top 10 2017: A5-Broken Access Control
- OWASP Top 10 2021: A10-Server-Side Request Forgery
- OWASP Top 10 API 2023: API7-Server Side Request Forgery

Result 1 of 1

High • Link • New • To Verify • Similarity Id: 941810963 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /auth_app/views.py	File Name: /auth_app/views.py
Method: fetch_url	Method: fetch_url
Element: GET	Element: get

Code Snippets

56 | url = request.GET.get('url')

57 | response = requests.get(url) # No URL validation

Path_Traversal (Type)

Query Path: Python/Python_High_Risk/Path_Traversal

CWE Id: 22

Total results: 1

Description: Method @SourceMethod at line @SourceLine of @SourceFile gets dynamic data from the @SourceElement element. This element's value then flows through the code and is eventually used in a file path for local disk access in @DestinationMethod at line @DestinationLine of @DestinationFile. This may cause a Path Traversal vulnerability.

Category:

- ASA Premium: ASA Premium
- Base Preset: Base Preset
- CWE top 25: CWE top 25
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Verification and representation of input data
- OWASP ASVS: V12 Files and Resources
- OWASP Top 10 2017: A5-Broken Access Control
- OWASP Top 10 2021: A1-Broken Access Control
- PCI DSS v3.2.1: PCI DSS (3.2.1) - 6.5.8 - Improper access control
- PCI DSS v4.0: PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development
- SANS top 25: SANS top 25
- Top Tier: Top Tier

Result 1 of 1

High • Link • New • To Verify • Similarity Id: 14630494 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /auth_app/views.py	File Name: /auth_app/views.py
Method: vulnerable_endpoint	Method: vulnerable_endpoint
Element: GET	Element: open

Code Snippets

82 | filename = request.GET.get('file')

83 | with open(f"/uploads/{filename}", 'r') as f:

Use_Of_Hardcoded_Password (Type)

Query Path: Python/Python_Medium_Threat/Use_Of_Hardcoded_Password

CWE Id: 259

Total results: 2

Description: The application uses the hard-coded password @SourceElement for authentication purposes, either using it to verify users' identities, or to access another remote system. This password at line @SourceLine of @SourceFile appears in the code, implying it is accessible to anyone with source code access, and cannot be changed without rebuilding the application.

Category:

- ASA Premium: ASA Premium
- ASD STIG 6.1: APSC-DV-001740 - CAT I The application must only store cryptographic representations of passwords.
- CWE top 25: CWE top 25
- FISMA 2014: Identification And Authentication
- MOIS(KISA) Secure Coding 2021: MOIS(KISA) Security Functions
- NIST SP 800-53: SC-28 Protection of Information at Rest (P1)
- OWASP ASVS: V02 Authentication
- OWASP Top 10 2021: A7-Identification and Authentication Failures
- OWASP Top 10 API: API2-Broken Authentication
- PCI DSS v4.0: PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development
- SANS top 25: SANS top 25

Result 1 of 2

Medium • Link • New • To Verify • Similarity Id: 857578285 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /vulnerable_app/settings.py	File Name: /vulnerable_app/settings.py
Method:	Method:
Element: "admin123"	Element: "admin123"

Code Snippets

50 | 'PASSWORD': 'admin123',

Result 2 of 2

Medium • Link • New • To Verify • Similarity Id: 1200348709 • Found First: 24 Jun, 2025 • Found Last: 24 Jun, 2025
First Scan ID: 2564dcc2-abc6-4b5d-b9be-7051cf129a9d

Source	Destination
File Name: /vulnerable_app/settings.py	File Name: /vulnerable_app/settings.py
Method:	Method:
Element: "hardcoded_secret_key_123"	Element: "hardcoded_secret_key_123"

Code Snippets

4 | SECRET_KEY = 'hardcoded_secret_key_123'

Missing_HSTS_Header (Type)

Query Path: [Python/Python_Medium_Threat/Missing_HSTS_Header](#)

CWE Id: [346](#)

Total results: 1

Description: The web-application does not define an HSTS header, leaving it vulnerable to attack.

Category:

- ASA Premium: [ASA Premium](#)
- OWASP ASVS: [V14 Configuration](#)
- OWASP Top 10 2021: [A7-Identification and Authentication Failures](#)
- OWASP Top 10 API 2023: [API8-Security Misconfiguration](#)
- PCI DSS v4.0: [PCI DSS \(4.0\) - 6.2.4 Vulnerabilities in software development](#)

Result 1 of 1

[Medium](#) • [Link](#) • [New](#) • [To Verify](#) • Similarity Id: [455451089](#) • Found First: [24 Jun, 2025](#) • Found Last: [24 Jun, 2025](#)
First Scan ID: [2564dcc2-abc6-4b5d-b9be-7051cf129a9d](#)

Source

File Name: [/vulnerable_app/settings.py](#)
Method:
Element: [CxPYNS_1df5e610](#)

Destination

File Name: [/vulnerable_app/settings.py](#)
Method:
Element: [CxPYNS_1df5e610](#)

Code Snippets

1 | `import os`

laC Vulnerabilities

No data to show

laC Scan Results (0)

No data to show

SCA Vulnerabilities

No data to show




SCA Scan results (0)




No data to show




SAST Resolved Vulnerabilities




No data to show




Categories

ASA Premium			
Category			
ASA Premium	8	5	3

ASD STIG 6.1			
Category			
APSC-DV-002510 - CAT I The application must protect from command injection.	1	0	0
APSC-DV-001740 - CAT I The application must only store cryptographic representations of passwords.	0	0	2
APSC-DV-002490 - CAT I The application must protect from Cross-Site Scripting (XSS) vulnerabilities.	2	3	0
APSC-DV-002540 - CAT I The application must not be vulnerable to SQL Injection.	5	0	0




Base Preset			
Category			
Base Preset	8	1	0

CWE top 25			
Category			
CWE top 25	8	5	2




FISMA 2014			
Category			
System And Information Integrity	8	4	0

Identification And Authentication	0	0	2
-----------------------------------	---	---	---




MOIS(KISA) Secure Coding 2021

Category			
MOIS(KISA) Verification and representation of input data	8	5	0
MOIS(KISA) Security Functions	0	0	2




NIST SP 800-53

Category			
SI-15 Information Output Filtering (P0)	2	3	0
SI-10 Information Input Validation (P1)	6	1	0
SC-28 Protection of Information at Rest (P1)	0	0	2

OWASP ASVS

Category			
V05 Validation, Sanitization and Encoding	7	4	0
V12 Files and Resources	0	1	0
V02 Authentication	0	0	2
V14 Configuration	0	0	1

OWASP Top 10 2013




Category			
A3-Cross-Site Scripting (XSS)	2	3	0

A1-Injection	6	0	0
--------------	---	---	---




OWASP Top 10 2017

Category			
A7-Cross-Site Scripting (XSS)	2	3	0
A5-Broken Access Control	0	2	0

OWASP Top 10 2021

Category			
A3-Injection	8	3	0
A10-Server-Side Request Forgery	0	1	0
A1-Broken Access Control	0	1	0
A7-Identification and Authentication Failures	0	0	3




OWASP Top 10 API




Category			
API2-Broken Authentication	0	0	2



OWASP Top 10 API 2023




Category			
API7-Server Side Request Forgery	0	1	0
API8-Security Misconfiguration	0	0	1

PCI DSS v3.2.1

Category			
PCI DSS (3.2.1) - 6.5.7 - Cross-site scripting (XSS)	2	3	0
PCI DSS (3.2.1) - 6.5.1 - Injection flaws - particularly SQL injection	6	0	0
PCI DSS (3.2.1) - 6.5.8 - Improper access control	0	1	0

PCI DSS v4.0			
Category			
PCI DSS (4.0) - 6.2.4 Vulnerabilities in software development	8	4	3

SANS top 25			
Category			
SANS top 25	7	4	2

Top Tier			
Category			
Top Tier	8	4	0

Stored_XSS (CWE 79)**What Is The Risk**

A successful XSS exploit would allow an attacker to rewrite web pages and insert malicious scripts which would alter the intended output. This could include HTML fragments, CSS styling rules, arbitrary JavaScript, or references to third party code. An attacker could use this to steal users' passwords, collect personal data such as credit card details, provide false information, or run malware. From the victim's point of view, this is performed by the genuine website, and the victim would blame the site for incurred damage. An attacker could use legitimate access to the application to submit modified data to the application's data-store. This would then be used to construct the returned web page, triggering the attack.

What Can Cause It

The application creates web pages that include untrusted data, whether from user input, the application's database, or from other external sources. The untrusted data is embedded directly in the page's HTML, causing the browser to display it as part of the web page. If the input includes HTML fragments or JavaScript, these are displayed too, and the user cannot tell that this is not the intended page. The vulnerability is the result of directly embedding arbitrary data without first encoding it in a format that would prevent the browser from treating it like HTML or code instead of plain text. In order to exploit this vulnerability, an attacker would load the malicious payload into the data-store, typically via regular forms on other web pages. Afterwards, the application reads this data from the data-store, and embeds it within the web page as displayed for another user.

General Recommendations

- * Fully encode all dynamic data, regardless of source, before embedding it in output.
 - * Encoding should be context-sensitive. For example:
 - * HTML encoding for HTML content
 - * HTML Attribute encoding for data output to attribute values
 - * JavaScript encoding for server-generated JavaScript
 - * It is recommended to use the platform-provided encoding functionality, or known security libraries for encoding output.
 - * Implement a Content Security Policy (CSP) with explicit whitelists for the application's resources only.
 - * As an extra layer of protection, validate all untrusted data, regardless of source (note this is not a replacement for encoding).

Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns. Check for:

- * Data type
- * Size
- * Range
- * Format
- * Expected values
- * In the `Content-Type` HTTP response header, explicitly define character encoding (charset) for the entire page.
- * Set the `HTTPOOnly` flag on the session cookie for "Defense in Depth", to prevent any successful XSS exploits from stealing the cookie.

SQL_Injection (CWE 89)**What Is The Risk**

An attacker could directly access all of the system's data. The attacker would likely be able to steal any sensitive information stored by the system, including private user information, credit card details, proprietary business data, and any other secret data. Likewise, the attacker could possibly modify or erase existing data, or even add new bogus data. In some scenarios, it may even be possible to execute code on the database. In addition to disclosing or altering confidential information directly, this vulnerability might also be used to achieve secondary effects, such as bypassing authentication, subverting security checks, or forging a data trail. Further increasing the likelihood of exploit is the fact that this flaw is easy for attackers to find, and easy to exploit.

What Can Cause It

The application stores and manages data in a database, by submitting a textual SQL query to the database engine for processing. The application creates the query by simple string concatenation, embedding untrusted data. However, there is no separation between data and code; furthermore, the embedded data is neither checked for data type validity nor subsequently sanitized. Thus, the untrusted data could contain SQL commands, or modify the intended query. The database would interpret the altered query and commands as if they originated from the application, and execute them accordingly. Note that an attacker can exploit this vulnerability either by modifying the URL, or by submitting malicious data in the user input or other request fields.

General Recommendations

- * Validate all untrusted data, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns.
 - * In particular, check for:
 - * Data type
 - * Size
 - * Range
 - * Format
 - * Expected values.
 - * Restrict access to database objects and functionality, according to the Principle of Least Privilege.
 - * Do not use dynamically concatenate strings to construct SQL queries.
 - * Prefer using DB Stored Procedures for all data access, instead of ad-hoc dynamic queries.
 - * Instead of unsafe string concatenation, use secure database components such as parameterized queries and object bindings (for example, commands and parameters).
 - * Alternatively, an even better solution is to use an ORM library, in order to pre-define and encapsulate the allowed commands enabled for the application, instead of dynamically accessing the database directly. In this way the code plane and data plane should be isolated from each other.

Command_Injection (CWE 77)

What Is The Risk

An attacker could run arbitrary system-level OS commands on the application server host. Depending on the application's OS permissions, these could include: * File actions (read / create / modify / delete) * Open a network connection to the attacker's server * Start and stop system services * Modify the running application * Complete server takeover

What Can Cause It

The application runs an OS system-level command to complete it's task, rather than via the application code. The command includes untrusted data, that may be controllable by an attacker. This untrusted string may contain malicious system-level commands engineered by an attacker, which could be executed as though the attacker were running commands directly on the application server. In this case, the application receives data from the user input, and passes it as a string to the Operating System. This unvalidated data is then executed by the OS as a system command, running with the same system privileges as the application.

General Recommendations

- * Refactor the code to avoid any direct shell command execution. Instead, use platform provided APIs or library calls.
 - * If it is impossible to remove the command execution, execute only static commands that do not include dynamic, user-controlled data.
 - * Validate all input, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified format, rather than rejecting bad patterns (blacklist). Parameters should be limited to an allowed character set, and non-validated input should be dropped. In addition to characters, check for:
 - * Data type
 - * Size
 - * Range
 - * Format
 - * Expected values
 - * In order to minimize damage as a measure of defense in depth, configure the application to run using a restricted user account that has no unnecessary OS privileges.
 - * If possible, isolate all OS commands to use a separate dedicated user account that has minimal privileges only for the specific commands and files used by the application, according to the Principle of Least Privilege.
 - * If absolutely necessary to call a system command or execute an external program with user input, do not use unsafe methods that call the system shell, such as ``os.system()`` or ``popen2.popen4()``.
 - * Instead, use safer methods such as ``subprocess.run()``, with the ``shell`` parameter set to ``False``.
 - * Always pass the user input as the 2nd element in the ``args`` list, with the first element in the list set to a hard-coded (or application-controlled) system command or program path.
 - * Do not pass the user argument as a string parameter, or as the first element in the ``args`` list.

Reflected_XSS (CWE 79)

What Is The Risk

A successful XSS exploit would allow an attacker to rewrite web pages and insert malicious scripts which would alter the intended output. This could include HTML fragments, CSS styling rules, arbitrary JavaScript, or references to third party code. An attacker could use this to steal users' passwords, collect personal data such as credit card details, provide false information, or run malware. From the victim's point of view, this is performed by the genuine website, and the victim would blame the site for incurred damage. The attacker could use social engineering to cause the user to send the website modified input, which will be returned in the requested web page.

What Can Cause It

The application creates web pages that include untrusted data, whether from user input, the application's database, or from other external sources. The untrusted data is embedded directly in the page's HTML, causing the browser to display it as part of the web page. If the input includes HTML fragments or JavaScript, these are displayed too, and the user cannot tell that this is not the intended page. The vulnerability is the result of directly embedding arbitrary data without first encoding it in a format that would prevent the browser from treating it like HTML or code instead of plain text. Note that an attacker can exploit this vulnerability either by modifying the URL, or by submitting malicious data in the user input or other request fields.

General Recommendations

- * Fully encode all dynamic data, regardless of source, before embedding it in output.
 - * Encoding should be context-sensitive. For example:
 - * HTML encoding for HTML content
 - * HTML Attribute encoding for data output to attribute values
 - * JavaScript encoding for server-generated JavaScript
 - * It is recommended to use the platform-provided encoding functionality, or known security libraries for encoding output.
 - * Implement a Content Security Policy (CSP) with explicit whitelists for the application's resources only.
 - * As an extra layer of protection, validate all untrusted data, regardless of source (note this is not a replacement for encoding).

Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns. Check for:

- * Data type
- * Size
- * Range
- * Format
- * Expected values
- * In the `Content-Type` HTTP response header, explicitly define character encoding (charset) for the entire page.
- * Set the `HTTPOOnly` flag on the session cookie for "Defense in Depth", to prevent any successful XSS exploits from stealing the cookie.

SSRF (CWE 918)

What Is The Risk

An attacker can abuse this flaw to make arbitrary requests, originating from the application server. This can be exploited to scan internal services; proxy attacks into a protected network; bypass network controls; download unauthorized files; access internal services and management interfaces; and possibly control the contents of requests and even steal server credentials.

What Can Cause It

The application accepts a URL (or other data) from the user, and uses this to make a request to another remote server. However, the attacker can inject an arbitrary URL into the request, causing the application to connect to any server the attacker wants. Thus, the attacker can abuse the application to gain access to services that would not otherwise be accessible, and cause the request to ostensibly originate from the application server.

General Recommendations

- * Do not connect to arbitrary services based on user input.
 - * If possible, the application should have the user's browser retrieve the desired information directly.
 - * If it is necessary for the application to proxy the request on the server, explicitly whitelist the allowed target URLs, and do not include any sensitive server information.

Path_Traversal (CWE 22)

What Is The Risk

An attacker could define arbitrary file path for the application to use, potentially leading to: * Stealing sensitive files, such as configuration or system files * Overwriting files such as program binaries, configuration files, or system files * Deleting critical files, causing denial of service (DoS).

What Can Cause It

The application uses user input in the file path for accessing files on the application server's local disk.

General Recommendations

1. Ideally, avoid depending on dynamic data for file selection.
2. Validate all input, regardless of source. Validation should be based on a whitelist: accept only data fitting a specified structure, rather than reject bad patterns. Check for:
 - * Data type
 - * Size
 - * Range
 - * Format
 - * Expected values
4. Accept dynamic data only for the filename, not for the path and folders.
5. Ensure that file path is fully canonicalized.
6. Explicitly limit the application to use a designated folder that is separate from the applications binary folder.
7. Restrict the privileges of the application's OS user to necessary files and folders. The application should not be able to write to the application binary folder, and should not read anything outside of the application folder and data folder.

Use_Of_Hardcoded_Password (CWE 259)

What Is The Risk

Hardcoded passwords expose the application to password leakage. If an attacker gains access to the source code, she will be able to steal the embedded passwords, and use them to impersonate a valid user. This could include impersonating end users to the application, or impersonating the application to a remote system, such as a database or a remote web service. Once the attacker succeeds in impersonating the user or application, she will have full access to the system, and be able to do anything the impersonated identity could do.

What Can Cause It

The application codebase has string literal passwords embedded in the source code. This hardcoded value is used either to compare to user-provided credentials, or to authenticate downstream to a remote system (such as a database or a remote web service). An attacker only needs to gain access to the source code to reveal the hardcoded password. Likewise, the attacker can reverse engineer the compiled application binaries, and easily retrieve the embedded password. Once found, the attacker can easily use the password in impersonation attacks, either directly on the application or to the remote system. Furthermore, once stolen, this password cannot be easily changed to prevent further misuse, unless a new version of the application is compiled. Moreover, if this application is distributed to numerous systems, stealing the password from one system automatically allows a class break in to all the deployed systems.

General Recommendations

- * Do not hardcode any secret data in source code, especially not passwords.
 - * In particular, user passwords should be stored in a database or directory service, and protected with a strong password hash (e.g. bcrypt, scrypt, PBKDF2, or Argon2). Do not compare user passwords with a hardcoded value.
 - * System passwords should be stored in a configuration file or the database, and protected with strong encryption (e.g. AES-256). Encryption keys should be securely managed, and not hardcoded.

Missing_HSTS_Header (CWE 346)

What Is The Risk

Failure to set an HSTS header and provide it with a reasonable "max-age" value of at least one year may leave users vulnerable to Man-in-the-Middle attacks.

What Can Cause It

Many users browse to websites by simply typing the domain name into the address bar, without the protocol prefix. The browser will automatically assume that the user's intended protocol is HTTP, instead of the encrypted HTTPS protocol. When this initial request is made, an attacker can perform a Man-in-the-Middle attack and manipulate it to redirect users to a malicious web-site of the attacker's choosing. To protect the user from such an occurrence, the HTTP Strict Transport Security (HSTS) header instructs the user's browser to disallow use of an unsecure HTTP connection to the the domain associated with the HSTS header. Once a browser that supports the HSTS feature has visited a web-site and the header was set, it will no longer allow communicating with the domain over an HTTP connection. Once an HSTS header was issued for a specific website, the browser is also instructed to prevent users from manually overriding and accepting an untrusted SSL certificate for as long as the "max-age" value still applies. The recommended "max-age" value is for at least one year in seconds, or 31536000.

General Recommendations

- * Before setting the HSTS header - consider the implications it may have:
 - * Forcing HTTPS will prevent any future use of HTTP, which could hinder some testing
 - * Disabling HSTS is not trivial, as once it is disabled on the site, it must also be disabled on the browser
- * Set the HSTS header either explicitly within application code, or using web-server configurations.
- * Ensure the "max-age" value for HSTS headers is set to 31536000 to ensure HSTS is strictly enforced for at least one year.
- * Include the "includeSubDomains" to maximize HSTS coverage, and ensure HSTS is enforced on all sub-domains under the current domain
 - * Note that this may prevent secure browser access to any sub-domains that utilize HTTP; however, use of HTTP is very severe and highly discouraged, even for websites that do not contain any sensitive information, as their contents can still be tampered via Man-in-the-Middle attacks to phish users under the HTTP domain.
- * Once HSTS has been enforced, submit the web-application's address to an HSTS preload list - this will ensure that, even if a client is accessing the web-application for the first time (implying HSTS has not yet been set by the web-application), a browser that respects the HSTS preload list would still treat the web-application as if it had already issued an HSTS header. Note that this requires the server to have a trusted SSL certificate, and issue an HSTS header with a maxAge of 1 year (31536000)
- * Note that this query is designed to return one result per application. This means that if more than one vulnerable response without an HSTS header is identified, only the first identified instance of this issue will be highlighted as a result. If a misconfigured instance of HSTS is identified (has a short lifespan, or is missing the "includeSubDomains" flag), that result will be flagged. Since HSTS is required to be enforced across the entire application to be considered a secure deployment of HSTS functionality, fixing this issue only where the query highlights this result is likely to produce subsequent results in other sections of the application; therefore, when adding this header via code, ensure it is uniformly deployed across the entire application. If this header is added via configuration, ensure that this configuration applies to the entire application.
- * Note that misconfigured HSTS headers that do not contain the recommended max-age value of at least one year or the "includeSubDomains" flag will still return a result for a missing HSTS header.

