University of Twente

Master Thesis

Threat Analysis of RPKI Relying Party Software

Koen van Hove

Supervisor:
prof. dr. ir. Roland van Rijswijk-Deij

Committee:
prof. dr. ir. Roland van Rijswijk-Deij
dr. ir. Andrea Continella
dr. Jeroen van der Ham

Faculty:
Electrical Engineering, Mathematics and Computer Science (EEMCS)

Chair:
Design and Analysis of Communication Systems (DACS)

2 February 2022

Chapter 1

Introduction

In agreement with the graduation committee, it was decided to document the work done during this Master of Science Thesis in the form of a paper. This paper will be submitted to the *USENIX Security '22* conference, which can be found in appendix A.

This paper fulfils the requirements as set by the examination board [1]. In the remainder of this document I show how I meet all the requirements for a Master of Science Thesis, as well as include the final thesis prior to outside modification.

[1]https://www.utwente.nl/en/eemcs/ids/education/assessment-msc/

Chapter 2

Requirements

Below I describe for each of the set requirements how they were met during the creation of this paper.

2.1 Scientific quality

2.1.1 Interpret a possibly general project proposal and translate it to more concrete research questions

The original project proposal was "create a threat model for the RPKI". This proposal started my literature research into the RPKI. After drawing up all the connections between different parts of the RPKI, it quickly became apparent that the relationship between certificate authority (CA) and relying party (RP), with the CA as malicious party, was an area of research that had not been widely explored. The RPKI Repository Delta Protocol (RRDP) was a new protocol specifically made for the RPKI, and in the discussion of its security aspects, nothing regarding this CA-RP relationship was mentioned. The lack of literature surrounding this attack vector inspired me to the following hypothesis: *it is probably possible to disrupt RP software as a CA using RRDP.* In order for me to better understand how the RPKI works from a CA perspective, I started creating my own CA software. During this process I accidentally made my RP software crash, which made my suspicions even greater.

This led to the following research question: What threats can an dishonest repository and/or certificate authority pose to relying party software with the aim of disrupting RPKI route validation using RRDP? To answer this question, I considered the following sub-questions:

1. Are there any exploitable security vulnerabilities caused by omissions

in the RFCs in the implementation specifications for RRDP?

2. What known threats exist to the stack that RRDP runs on?
3. What unspecified or explicitly allowed aspects of RPKI/RRDP can a repository and/or certificate authority abuse to make the relying party software malfunction?

The questions are set up in such a way that they support the overall research question, as well as ensure that it is an as complete as possible view of the CA-RP landscape. In the final paper, the questions have been condensed, and the research has expanded beyond purely RRDP, and also considers the previous protocol used: rsync. The answers to the original questions can still be found in the paper.

2.1.2 Find and study relevant literature, software and hardware tools, and critically assess their merits

As mentioned in the previous section, formulating the research questions required me to innately understand the RPKI, which meant researching the literature of how it works, how it came to be, and what research has been conducted already. For one, I first learned of the RPKI when visiting RIPE 78 in 2019, but apart from a rough understanding of “securing BGP”, I did not know any of the details.

2.1.3 Work in a systematic way and document your findings as you progress

The deadline was set as 1 February 2022 by USENIX Security ’22. This provided a hard deadline for the paper. Furthermore, the resulting findings needed to be communicated with the National Cyber Security Centre of the Netherlands (NCSC) and the initially four organisations in the context of a coordinated vulnerability disclosure (CVD). The reason for this was that during the research some vulnerabilities were found that could easily be exploited in the wild, which would have a large (inter)national impact on the entire RPKI ecosystem. This collaboration between the organisations and the University of Twente required a high level of systematisation and proper documenting of the findings in order to provide the organisations with adequate information to resolve the issues at hand.

2.1.4 Work in correspondence with the level of the elective courses you have followed

Many of the electives I have followed revolve around logic, networking, and security. What assumptions were made and whether those assumptions hold was for example a core part of the *Distributed Systems* elective. Furthermore, I want to highlight that ethics played a large part in this final project due to the CVD. As this paper will be submitted to an actual conference, I believe the level of work meets the level expected of a Master course.

2.1.5 Perform original work that has sufficient depth to be relevant to the research in the chair

As mentioned previously, as my supervisor considers the work suitable for submission to a conference in my opinion shows that the work is 1) original and novel; and 2) has sufficient depth.

2.2 Organisation, planning, collaboration

2.2.1 Work independently and goal oriented under the guidance of a supervisor

The majority of the work was done by me. I took the initiative for the majority of the time, whilst keeping my supervisor informed of the actions I took. The CVD required careful coordination and communication between the University, the NCSC, and the organisations, which I did to a large extent.

2.2.2 Seek assistance within the research group or elsewhere, if required and beneficial for the project

As mentioned previously, during the project I worked together with the NCSC and other organisations such as the RIPE NCC. This was majorly beneficial to get an insight in how the RPKI works from an operational point of view, as well as understand what decisions were made based on technical merit, and which decisions were made based on politics. I also consulted with implementers from NLnet Labs, NIC.mx, and others to find the cause of the issues, and work together on solutions to the problems at hand.

2.2.3 Benefit from the guidance of your supervisor by scheduling regular meetings, provide the supervisor with progress reports and initiate topics that will be discussed

I regularly kept my supervisor up-to-date with my progress. We did not schedule regular meetings, as we deemed them unnecessary, but I did schedule meetings if I felt like there was something we needed to discuss.

2.2.4 Organize your work by making a project plan, executing it, adjusting it when necessary, handling unexpected developments and finish within the allotted number of credits

As mentioned before, the final deadline was set at 1 February 2022. The project plan was made to easily accommodate this. The CVD, as well as the unexpected turns the CVD took, did mean certain things took longer than expected, and required some modification of the plan, but the final deadline was never at risk. The CVD timeline can be found in appendix A of appendix A.

2.3 Communication

2.3.1 Write a Master thesis that motivates your work for a general audience, and communicates the work and its results in a clear, well-structured way to your peers

I have, with permission, a paper, thus the target audience is my fellow peers at the conference. As this is a paper that will be submitted to the USENIX Security '22 conference, I believe this ensures the work and its results are communicated in a clear, well-structured way.

2.3.2 Give a presentation with similar qualities to fellow-students and members of the chair

This work has been presented at the University, as well as at APRICOT 2022.

Appendix A

USENIX Security '22

The appendix contains the final version of the conference paper *rpkiller: Threat Analysis from an RPKI Relying Party Perspective* which was written before alterations by any of the other authors. The acceptance rate was 16.1% in 2020[1].

Name	USENIX Security '22
Venue	Boston, MA, United States of America
Web page	https://www.usenix.org/conference/usenixsecurity22
Date	August 10-12, 2022
Submission deadline	February 1, 2022
Notification date	May 2, 2022
Final version due	June 14, 2022

[1]https://www.usenix.org/sites/default/files/sec20_message.pdf

rpkiller: Threat Analysis from an RPKI Relying Party Perspective

Koen van Hove
University of Twente

Abstract

The Resource Public Key Infrastructure (RPKI) is a framework that aims to secure routing by creating an infrastructure where resource holder can attest statements about their resources. Certificate Authorities (CAs) publish these statements at publication points. Relying party software retrieves and processes the RPKI-related data from all publication points. It does so using the RPKI Repository Delta Protocol (RRDP) – a protocol based on XML and HTTPS – and rsync. We create a threat model for relying party software, where an attacker controls a certificate authority and publication point. We analyse with a prototype how current relying party software reacts to scenarios originating from that threat model. Results show that all current relying party software was susceptible to at least one of those threats, with some threats stemming from choices made in the protocol itself, allowing us to fully disrupt RPKI relying party software on a global scale.

1 Introduction

The internet consists of *inter*connected *net*works managed by many different organisations. Data is exchanged within these networks, but also between these networks, e.g., data from Google to an internet customer at Comcast. Just like addressing physical mail, the data on the internet needs an address. This address is generally an Internet Protocol (IP) address, of which two versions are currently in wide use, namely IPv4 and IPv6. Much like physical mail, there are arrangements between operators to handle each other's data. The Border Gateway Protocol (BGP) is a protocol created to coordinate this. Every network, also called an Autonomous System (AS), sends 1. who they are; 2. who can be reached (indirectly) via them. They send this to the peers they are connected to. Every network has an Autonomous System Number (ASN). By creating a list of these ASNs, one can eventually discover how to reach all destinations. This string of numbers is also called an AS path, and allows every network to find a path to any other network [59].

The Resource Public Key Infrastructure, or RPKI for short, is a public key infrastructure framework that aims to work together with BGP [27]. RPKI tries to secure routing in the same way that WebPKI tries to secure the connection between your web browser and the web server, where a certificate attests that you are actually talking to the website you intended to visit. BGP forms an integral part of the worldwide inter-AS routing system, and is not easily replaced. The main issue facing BGP currently is that it is largely based on good faith. By default, anyone can send any BGP announcement, with their own ASN as the destination for *any* prefix [18]. This means someone else than the owner is now in control of a set of IP addresses, and can use them for their own purposes. These situations are not purely hypothetical - they have occurred in the past due to misconfiguration or malice [19, 65].

RPKI currently solves that issue by providing attestation using aforementioned certificates. There are five generally agreed upon certificate trust anchors, namely the five Regional Internet Registries (RIRs) – AfriNIC, APNIC, ARIN, LACNIC, and RIPE. These five RIRs all host Route Origin Authorizations (ROAs) for their customers directly in their own repository, but it can also be delegated, where customers host their own repository. Relying party software retrieves the data from the repositories recursively, and then processes it. Deployment is increasing rapidly, especially when it comes to delegated RPKI. A question that arises is whether a dishonest repository owner can abuse their power to disrupt relying party software, and potentially RPKI as a whole.

For that reason, our research question is: **How can a dishonest repository or certificate authority disrupt the operations of relying party software?** To answer this question, we consider the following sub-questions:

1. What known threats exist to the stack that the RPKI runs on?

2. Are there any exploitable security vulnerabilities caused by omissions in the RFCs in the implementation specifications?

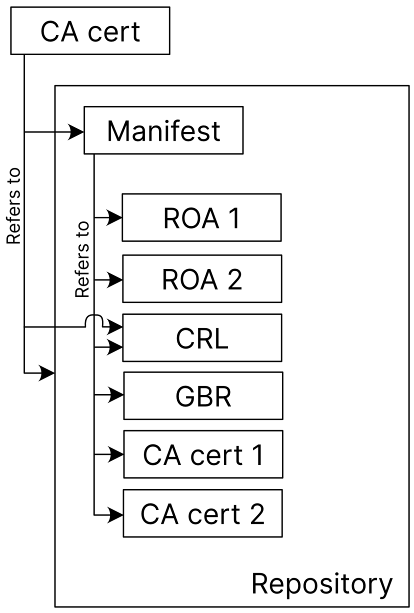


Figure 1: An example of a repository in RPKI. The CA cert contains an extension that states where the repository can be found, and what file inside the repository the manifest (MFT) is. The manifest then contains entries to the ROAs, Certificate Revocation List (CRL), Ghostbusters Record (GBR), and child CA certs. The objects in the repository (MFT, ROA, CRL, GBR, and CA certs) are CMS signed objects [20] using a certificate signed by the CA.

3. What unspecified or explicitly allowed aspects of RPKI can a repository and/or certificate authority abuse to make the relying party software malfunction?

To answer these questions, we will:

1. Develop a threat model for RPKI focussing on the abilities of a dishonest repository and/or certificate authority with identified possible threats;

2. Create a proof-of-concept exploit framework and an overview of the behaviour of relying party software with regards to these situations.

The remainder of this paper is organised as follows: in section 2, we outline the background of RPKI, and our considerations for focussing on this particular attack vector. In section 3 we describe the related work. In section 4 we create a threat model. In section 5 we create real test cases based on the threat model and test those on real relying party implementations. In section 6 we discuss these results, and what a solution could look like. Lastly, in section 7 we describe the process we used to notify the parties of these vulnerabilities.

2 Background

2.1 BGP

The Border Gateway Protocol (BGP) is a protocol that aims to exchange routing and reachability information. BGP generally comes in two varieties: interior (within an AS) and exterior (between ASes). We only concern ourselves with the latter – an AS has full control over the routing within their network, and is thus out of scope.

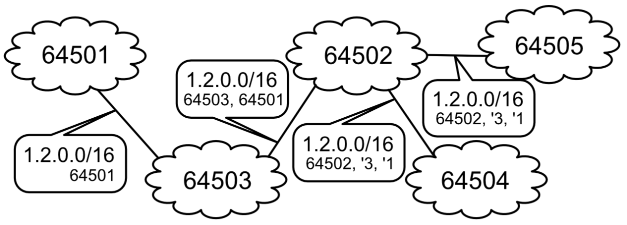


Figure 2: An example of a network with five nodes, 64501 to 64505, with lines representing the connections. An example of the BGP advertisement for 1.2.0.0/16 from 64501 is shown.

Let us explain how BGP works by means of example as shown in figure 2. The nodes 64501 to 64505 represent our ASes, and the dashed lines represent connections between them. Initially, the nodes do not know about each other. AS 64501 can advertise to 64503 that 64501 is the destination for 1.2.0.0/16, thus an AS path of $\langle 64501 \rangle$. 64503 learns that information, and advertises to 64502 (and potentially to 64501) that 1.2.0.0/16 can be reached via $\langle 64503, 64501 \rangle$. Repeat this, and 64504 learns that a path to 1.2.0.0/16 is $\langle 64502, 64503, 64501 \rangle$. This is depicted in figure 2. If 64505 also advertises 1.2.0.0/16, then 64504 will learn that $\langle 64502, 64505 \rangle$ is also a route for that prefix. Generally a shorter route is preferred. BGP uses the principle of longest prefix matching, meaning if a more specific prefix were to be advertised, e.g., 64505 advertising 1.2.3.0/24, then 64503 would generally prefer that over the more general announcement of 1.2.0.0/16 from 64501. Note that in practice every AS has its own policy for routing, for example regarding the maximum prefix length. BGP merely makes claims about what can be reached, not which path should be used. An operator may thus pick a different path.

1.2.0.0/16 does not appear in the clouds in figure 2. The reason for this is that the claim by both 64501 and 64505 is entirely based on good faith. BGP does not provide any means to verify any of the claims made. It is thus possible to claim to be the destination of a prefix owned by someone else, or to claim a different AS is the destination, or to claim that an AS has connections to other ASes which it does not have. For example, 64505 may advertise a path of $\langle 64505, 64501 \rangle$ for 203.0.113.0/24, even though that does not exist in real life and 64501 is not the destination for that prefix.

2.2 Increasing BGP security

There have been many attempts at making BGP more secure, such as BGPsec [30]. One of the main hindrances in most of them is that they require a fundamental change in the BGP protocol that is backwards incompatible with the current version, thus requiring a flag day. RPKI is a separate infrastructure that can be implemented gradually, and thus avoids the issue of requiring a flag day. RPKI aims to enable creating attestable statements about internet number resources, but in its currently only contains something called Route Origin Authorisations, or ROAs for short [29]. A ROA contains an ASN and one or more IP prefixes. If one then receives an announcement for an IP prefix, one can lookup whether there exists a ROA that covers this prefix. The result can be: 1. valid - it exists, and the ASN matches the ASN of the ROA; 2. invalid – it exists, and the ASN does not match the ASN of the ROA, or the prefix length does not match the length of the ROA; 3. unknown (or “not found") – there is no ROA for this IP prefix [22]. This solves part of the good faith issue described above. It is important to note that a ROA only aims to protect the final destination for a BGP path, meaning that a valid destination does not guarantee an honest path. RPKI may be used in the future for this as well, with for example ASPA [3]. Additionally, it is to the recipient’s full discretion what to do with the information from RPKI. They are free to ignore it entirely or partially.

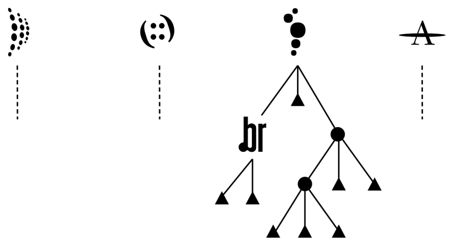


Figure 3: An example of a tree showing trust hierarchy in RPKI. The example shows an example tree originating from LACNIC. Every circle, as well as the RIRs and registro.br (a NIR from Brazil), represent a CA. Every triangle represents a ROA. Every subordinate CA can delegate (part of) their INR to further child CAs. The five trees of the five RIRs are in principle independent.

To determine whether a ROA is genuine, a standard PKI setup using X.509 certificates [6], with signing and certificate trees is used. There are five generally agreed upon certificate trust anchors, namely the five Regional Internet Registries (RIRs) – AfriNIC, APNIC, ARIN, LACNIC, and RIPE (although one is technically able to choose their own trust anchors, but just like WebPKI this is mostly hypothetical) [27]. These five RIRs all host ROAs for their customers directly, but they may offload tasks to subordinate certificate authorities, such as National Internet Registries (NIRs), or organisations that prefer to host their own ROAs. These subordinate certificate authorities can again create their own subordinate certificate authorities, etc. They also host a repository, which is a place where the ROAs, certificates, and other objects can be downloaded from. These repositories are referenced in the certificates [21]. Note that whilst normally a certificate authority and repository owner are managed by the same entity, this need not be the case. An example of a trust hierarchy tree can be seen in figure 3.

From a technical perspective, this is done in the following way: at the top of the tree are the five RIRs, each with their own root certificate and repositories. Those repositories, accessible via rsync or RRDP, a protocol based on HTTPS and XML, contain signed objects. Signed objects are currently mainly ROAs, although Ghostbusters records have been added as well [9]. These objects are signed with a certificate signed by the certificate authority. The repository also contains other certificates signed by the current certificate. Those certificates then point to their own repository, which is what gives it the tree structure. For every certificate, there is a manifest file. A manifest file contains a list of expected signed objects and their hash - this can be used to check that the data from the repository is complete and correct [21]. An example of a repository can be seen in figure 1. Do note that a single repository may host multiple manifests, and the CA certificates may refer to another manifest within the same repository.

Recently, more subordinate certificate authorities and repositories have appeared [25]. relying party software, the software that collects all the ROAs and creates the lookup table, traverses these subordinate certificate authorities and their repositories. Previously, all repositories belonged to trusted parties, such as regional internet registries, national internet registries, or a select few miscellaneous parties that were generally well-trusted.

2.3 CA and repository

It is important to differentiate between the notion of a repository and a CA:

Repository A repository, also called a publication point, is a place where the data from the RPKI can be found. A repository can be conceptualised as a “folder" with signed files (e.g. ROAs) in them.

Certificate Authority A certificate authority (CA) is the holder of the private key that can sign the files.

In a lot of cases the operator of the repository is also the operator of the certificate authority, but these can be two different entities. For example, a NIR can operate the repository, whilst the signing is done by the customers directly. Additionally, in the case where RIRs host the ROAs for their customers directly, it is often the case that the RIR is both the repository and certificate authority, and the customer can only specify what objects the RIR should create.

2.4 Rsync

The RPKI supports two protocols: rsync and RRDP. Initially only rsync was used [27], and support for rsync is, at the moment of writing, still required. All common relying party software now also supports RRDP, as do nearly all of the repositories. There are already several known issues associated with the usage of rsync that have spurred the adoption of RRDP. We will itemize a couple of them:

1. rsync is a protocol specified by its implementation. All current relying party implementations currently use the reference implementation of rsync, with the exception of rpki-client, which uses an open client-side reimplementation of the early 2004 version of the rsync protocol (version 27) called OpenRsync [56]. This adds a layer of difficulty for relying party software, as they now need to interact with an external rsync process.

2. rsync repositories are plagued by non-atomic updates [8, 66], meaning that if the content of a repository changes whilst a relying party is retrieving the data, that attempt will fail as the manifest entries do not match the retrieved data.

3. It is trivial to execute a successful denial-of-service attack on the rsync daemon by having a desktop open several thousand connections to an rsync daemon from a /48 IPv6 subnet, different Tor exit nodes, several VPN endpoints, or a combination of the former. This causes the rsync daemon to be overloaded, and bars honest parties from retrieving the data from the repository over the rsync protocol. This is against RFC 6481, which states that “The publication repository SHOULD be hosted on a highly available service and high-capacity publication platform." [21]. Due to the reference implementation nature of rsync, we consider resolving this issue infeasible.

4. rsync clients are easily disrupted, for example using a very large “message of the day", or serving a repository with millions of empty folders to cause inode exhaustion. This causes the relying party software to crash.

These are not merely theoretical, but have been confirmed in practice, and are well-known [7]. Given that, as well as the current uptake of RRDP, we expect rsync to be fully replaced in practice by RRDP by the end of 2022. As a consequence, the main consideration will be RRDP, although we may consider rsync as well if the issue transcends the protocol used.

3 Related work

The concept of having a system that recursively retrieves data from unknown servers is not new. An example of this is practice can be found in DNS [48]. One main difference between DNS and RPKI is that DNS needs to find one path to the record it requested, and can ignore all other information, whereas RPKI collects all information first before it builds a table. This means a malicious DNS entry for evil.example.org should not impact resolving benevolent.example.com. DNSSEC aims to improve DNS security, but at its inception did not specify what parts it tried to protect [2]. There have been external models checking the claims made about DNSSEC, such as done by Bau et al. [4]. However, it is clear that the threat model differs from ours, as our threat model mostly concerns disruption, something DNSSEC hardly touches upon, and Bau et al. do not take into consideration.

BGP, the system RPKI aims to protect, also works based on a table. However, unlike RPKI, this table is built incrementally whenever information comes in [59]. There are availability vulnerabilities in specific implementations, such as BIRD [35], but those cases appear isolated. Additionally, BGP does not contain the same hierarchy as RPKI does. BGPsec aims to improve BGP security [30], but much like DNS the focus in the threat model is on integrity and verifiability of the data, and not on availability.

TLS 1.3, the most recent TLS standard, does have a well-described security models. They however mostly focuses on confidentiality and non-tamperability [15, 60], and not on availability. Like BGP, isolated cases of availability vulnerabilities are present here as well [37].

To our knowledge RPKI is unique in the field of computer networking in that it creates a table based on a arbitrary hierarchical data structure, where data is retrieved from arbitrary non-trusted servers, where it needs all information first in order to function correctly.

In the field of RPKI, the focus has mostly been on the external effects the infrastructure provides, instead of looking into the security of the infrastructure itself. For example, Wählisch et al. [70] look at the deployment of RPKI, and what the reasons are deployment lags behind, and which attacker models RPKI prevents. There has been research into the security of RPKI itself, such as the importance of consistency of objects [28], the lifetime of objects and certificates [24], and the use of the maxLength attribute [17]. Cooper et al. [13] look at what power different entities within the RPKI ecosystem have, and what certain authorities could do to abuse that power. Shrishak et al. [64] also look at this, and come up with a solution that could be used to solve the issue that one RPKI authority may have too much power. Others have come up with solutions that involve the Blockchain [71] to counter the centralised nature of RPKI. Additionally, RFC 7132 [23] mentions some security considerations of the results of an adversary CA, e.g., “An attacker could create very deep subtrees with many ROAs per publication point, etc.", but it does not however assess the impact in any great detail, nor does it create a comprehensive analysis of threats to RPKI itself.

It seems implied that, although not stated explicitly, that this type of attack is from a purely theoretical nature due to the unlikelihood of a CA takeover, with an expectation that a CA can normally be trusted.

There is little public security research into RRDP. The RRDP RFC [8] does include security considerations, but those are mostly regarding the integrity and tamperproofing, which references the recommendations for the use of TLS [63], and some additional notes on how RRDP increases reliability. RRDP is however built upon XML and HTTPS, and we consider it worth looking into whether the known vulnerabilities of those technologies affect RRDP as well.

4 Threat model

For our threat model, we assume that an attacker has its own CA and repository in a non-malicious RPKI tree at the same level as other non-malicious organisations, which includes the ability to sign new CAs and objects. We will sketch the assumptions made about the RPKI in section 4.1, and then expand on the implications of that in section 4.2.

4.1 Assumptions

Let us look at the implicit assumptions made about the RPKI, and whether those assumptions are guarded by technological measures or as part of the standards. There are generally three types of assumptions made: tree size, time, and stack.

4.1.1 Tree size

When one considers the RPKI, the most common visualisation is a forest of trees as depicted in figure 4. The first assumption is thus that the RPKI is a forest of independent trees, where each tree stems from a Trust Anchor Locator (TAL), a reference to the root certificate preloaded into the relying party software. Each tree must be correctly evaluable without information from the other trees, and no loops must be possible. Furthermore, the RPKI presumes one first collects all information from the repositories before making routing decisions. For example, if a parent CA has a ROA for a /8, and a child has a ROA for a subset of that /8 (e.g., a /16) with a different ASN, and the child's repository is unavailable, then a route advertisement for that /16 might switch from valid to invalid (instead of unknown), which means routes become unavailable. If we take the tree for figure 4, and we suppose ε has a ROA for 1.0.0.0/8 with AS 1, and η has a ROA for 1.2.0.0/16 with AS 2. If a BGP advertisement comes in for 1.2.0.0/16 with a path that ends with AS 2, it is considered valid, as a known ROA for it exists. If we however do not consider the data from η, then that same advertisement would not match a known ROA, and instead, it would fall under the ROA from ε for 1.0.0.0/8 with AS 1, which would make this BGP advertisement invalid (rather than unknown).

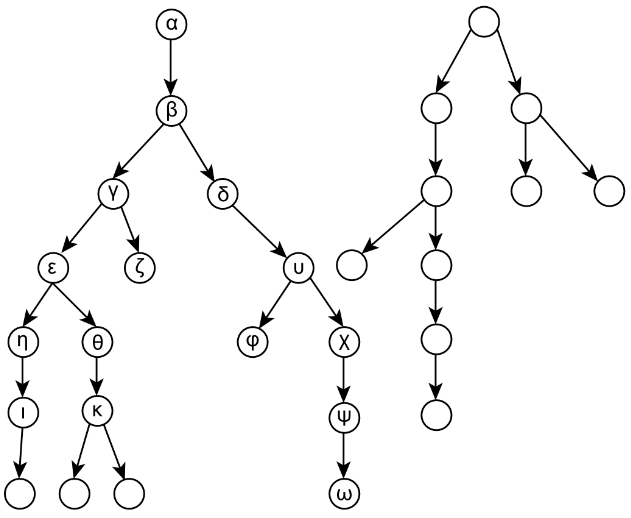


Figure 4: An example of two RPKI trees. Each arrow signifies a parent-child relationship.

When we look at the nodes themselves, we make another set of assumptions. Let us look at node υ as shown in figure 4. We assume that the behaviour of node υ may affect itself, as well as its children ϕ to ω, but that its behaviour may not influence the validity of its parents or its siblings, in this case α to κ. Furthermore, a node may not require information from nodes other than its parents to function, thus υ must be able to work with only information from α, β and δ. It is also assumed that the view I have is the same as the view others have of the RPKI, and that no different data is returned based on the origin.

This means that the assumption is made that the tree is finite and reasonably sized, and that that is the case for everyone.

4.1.2 Time

All these assumptions about the tree size are there for one main reason: the certificates and objects in the RPKI all have an expiration time after which they are no longer valid. It is thus also assumed that the tree can be evaluated and processed in reasonable time, generally considered to be between 30 and 60 minutes [25]. If the process takes significantly longer, objects will have expired before they could be used. What is "too long" is not defined, but the shortest objects have been observed in practice to have a lifetime of around 8 hours.

4.1.3 Stack

As stated before, RRDP is a protocol that is based on XML and HTTPS. This means that the threats that apply to that stack, namely XML, HTTPS, and TCP (or UDP in the case of HTTP/3 [5]) apply to RRDP as well. RRDP applies some extra restrictions, such as requiring US-ASCII as character set, and forbidding the verification of the WebPKI certificate [8]. The signed objects themselves are CMS-encoded X.509 objects [20], which means the threats to signed objects, certificates, and ASN.1 validation apply here as well. The only file in a repository that is not a signed object is a certificate to another repository in the tree. Lastly, the relying party software runs on a server, which most likely runs a UNIX-based operating system. This means that OS threats, such as inode exhaustion, running out of disk space, memory, or CPU cycles apply here as well. This list is not exhaustive.

4.2 Implications

4.2.1 XML

Several XML exploits exist, such as 1. coercive parsing, where an extreme (possibly endless) depth file is parsed; 2. invalid structures, where the XML parser may trip up due to the unconventional nature of the file; 3. very large payloads, where the XML parser may load everything into memory, or temporarily store them on disk, causing resource exhaustion; 4. using external entities, thereby requesting resources it should not be able to access, and possibly forwarding those to a remote server; 5. entity expansion, by specifying recursive entities, or referential entities, an XML parser may run out of space or memory; 6. using external references in XML to let the client DDoS another entity. [58]

4.2.2 HTTPS

Several HTTPS exploits exist, such as 1. decompression bombs, where if the client supports compression such as GZIP, the server may serve a small file that is orders of magnitudes larger when unpacked [11]; 2. returning a 301 code with a location header with a location that either loops, or keeps forwarding to another location; 3. returning a 429 code with a retry-after header with an unreasonably high value, such as a year or century. [57]

4.2.3 TCP

Attacks on TCP exist. These mostly concern on stalling the connection, for example by a Slowloris attack [12], or simulating a very slow unreliable connection.

4.2.4 X.509

X.509 is a very elaborate protocol, of which RPKI only allows a subset. However, many relying parties use a standard library for X.509 that supports far more than RPKI requires. This makes it vulnerable to attacks such as specially crafted private keys [34]. Additionally, given the binary nature of this format, unexpected data may cause the application to misbehave [33].

4.2.5 ASN.1

ASN.1 is a binary format that needs to be decoded, which in the past has had vulnerabilities in some major implementations like OpenSSL [32] and BouncyCastle [36]. Additionally, RPKI defines some uncommon modules with extra constraints, such as decoding an IP address prefix as BITSTRING, which may not be adequately checked, and where strange values may cause undefined behaviour.

4.2.6 Trees

RPKI repositories are assumed to be structured like a finite tree, where the application goes past every repository. This is similar to directory traversal, and several exploits exist here as well. A repository can for example introduce loops, or create an endless depth chain of certificates, thereby causing the relying party to endlessly fetch data. One can also introduce very many children, and thus go wide instead of deep. One can also combine the two.

4.2.7 Operating system

One can also try to target the underlying file system. Many Linux systems use ext4 as filesystem, which has a limited amount of inodes (pointers to entries) available. One can thus serve a repository with many objects with the aim to exhaust the amount of inodes. One may also try to serve extremely large files, like a several gigabyte large Ghostbusters record to exhaust disk space [14].

5 Evaluating threats in practice

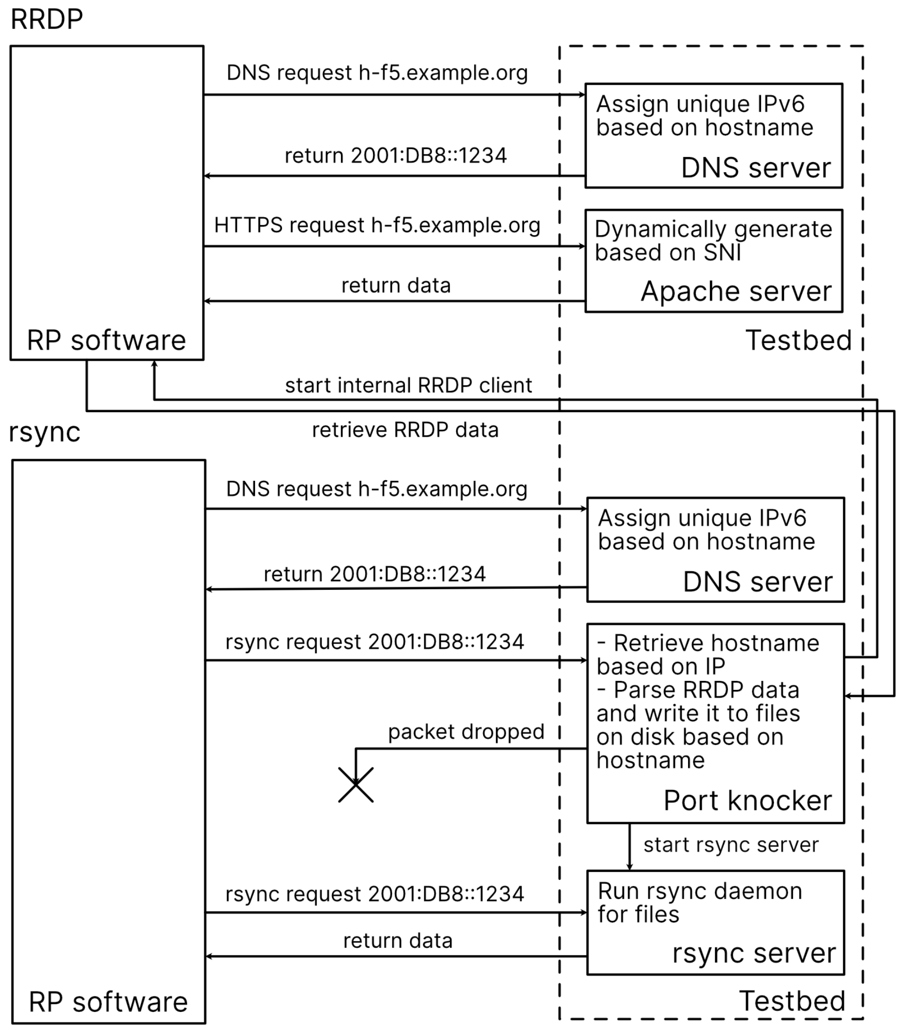
Based on the analysis in section 4, we identified a set of potential vulnerabilities. To test if current relying party implementations were susceptible to these vulnerabilities, we designed and implemented a testbed that instantiates these vulnerabilities. The tests are labelled A to O, ordered chronologically.

5.1 Testbed

The testbed creates custom repositories and CAs, and a separate TAL is available for each test instance. The server creates the files for each instance on-demand using a randomly generated UUID version 4 [26], meaning that two test instances cannot influence each other, and the result is not influenced by caching. The resulting URI looks like `h-f5654a8f-6d17-4b62-9bb8-d5e11b8c08b2.example.org`, where the first letter of the hostname indicates the test (in this case H), and the part after the first dash is the identifier. This host is assigned its own unique IPv6 address, and serves all requests for this test instance of test H over both RRDP and rsync. The full overview of the setup can be found in figure 5.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Routinator	●			●	●			●		●	●			●	
Validator 3	●	☒	☒	☒	☒	☒	☒	●	☒	☒	☒	☒	☒	☒	☒
OctoRPKI	●			●	●	●		●	●	●	●	●		●	●
Fort					●			●		●	●	●			●
RPKI-Prover	●						●	●		●	●				
rpstir2	●			●	●	●		●	●	●	●	●		●	●
rpki-client					●			●		●	●	●			
rcynic	☒	☒	☒	●	☒	☒	☒	●	☒	☒	☒	☒	☒	☒	☒

Table 1: The tests executed during our research in the then-recent relying party software. ● means an implementation was vulnerable. The crossed-out cells for Validator 3 and rcynic were not tested.



RRDP
DNS request h-f5.example.org
Assign unique IPv6 based on hostname
DNS server
return 2001:DB8::1234
HTTPS request h-f5.example.org
Dynamically generate based on SNI
Apache server
return data
RP software
Testbed
start internal RRDP client
retrieve RRDP data
rsync
DNS request h-f5.example.org
Assign unique IPv6 based on hostname
DNS server
return 2001:DB8::1234
rsync request 2001:DB8::1234
- Retrieve hostname based on IP
- Parse RRDP data and write it to files on disk based on hostname
Port knocker
packet dropped
start rsync server
rsync request 2001:DB8::1234
Run rsync daemon for files
rsync server
return data
RP software
Testbed

Figure 5: A graph overview of how the testbed works, both for RRDP and rsync. For RRDP, the Server Name Indication (SNI) [1] extension is used, which contains the hostname that was requested. As this extension (or similar) is absent for rsync, the IP address used in the request is converted back to the corresponding hostname, and the testbed itself emulates the RRDP request. It then converts the RRDP output to files (if possible), and starts the rsync daemon listening on that IP address. The original request packet is dropped, and the relying party (RP) software retries after a timeout, which request is served directly by the rsync server.

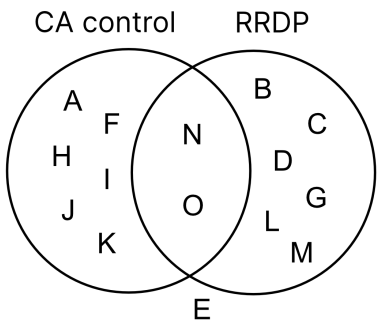


Figure 6: The level of control required to execute a specific attack. For CA control, the attacker needs to be able to sign new objects. For RRDP, the attack only works via an RRDP exploit, and not over rsync.

All implementations ran on their own virtual machine, with only the implementation installed according to the instruction manual provided by the implementation. The specifications of the virtual machines used is listed in table 2. A partial description of these tests was also published on RIPE Labs [67]. The RIPE NCC RPKI Validator 3 and rcynic were not fully tested due to them no longer being actively supported. The level of access an attacker would need to successfully execute the attack is shown in figure 6.

5.2 Tests

The success of our tests is shown in table 1. Relying party software is constantly being updated; this research is thus not meant as a definitive analysis of current relying party software, but rather as an overview and threat model of dishonest RPKI repositories and/or certificate authorities, with the aim of hardening current and future relying party software. It thus might be that threats mentioned in this research do not impact current relying party software, but might impact future relying party software that adheres to the RFCs.

5.2.1 (A) Infinite repository chain

We create a chain of repositories, where the repository root has a certificate for its child, and when that child is visited, a certificate for its child is generated. This is repeated ad infinitum, creating an infinite repository chain. This tries to attack the assumption that the tree is finite, and makes it that clients get stuck retrieving new data forever.

Most tested implementations keep retrieving data endlessly. Initial tests showed only Fort and rpki-client limit the maximum depth of a repository. This changed later as Routinator also added a maximum depth.

5.2.2 (B) 429 response header

The 429 HTTP status code was added in RFC 6585 [16]. It specifies that a user has sent too many requests, and should apply rate limiting. To provide a hint to the user when they can try it again, a “Retry-After" may be included which contains the amount of seconds a user should wait before trying it again. RFC 8182 [8] does not specify whether this HTTP status code can appear in RRDP, and as relying party software tends to use existing HTTP libraries that might abstract this away, we wanted to find out what would happen if this value was set to an unreasonably long duration, for example a day. Older implementations sometimes instead use the 503 status code with Retry-After header.

All tested implementations do not support 429 rate limiting, and neither do the libraries that they use.

5.2.3 (C) Endless 302 response

The 3xx range of status codes indicate that the resource can be found at another location [53]. These status codes can be chained, i.e. A can link to B which links to C. Much like test A, this can be done ad infinitum, thereby endlessly redirecting without technically looping. Relying party software is not required to support 3xx status codes, nor is limiting the maximum amount of redirects required. In most cases the default value from the HTTP library is used.

All tested implementations either do not support 3xx redirects, or if they do, they limit it to a small amount, generally below 10.

5.2.4 (D) GZIP bomb

HTTP allows for compression to improve transfer speed [54]. This means that the amount of data transferred may be less than the unzipped size on the disk. Gzip is such a compression algorithm, and it is possible to create a file that is orders of magnitudes larger uncompressed than it is compressed, thereby exhausting the memory of the system the relying party software runs on. Support for compression is generally explicitly enabled in the used HTTP libraries.

Routinator, OctoRPKI, rpstir2, and rcynic crash due to being out of memory, whereas Fort, rpki-client, and RPKI-Prover reject the repository.

5.2.5 (E) Open connection

Repository provide data at a certain bandwidth – a minimum bandwidth is required to adhere to the reasonable time requirement. It is possible to break this bandwidth assumption by limiting it to, for example, 3 bytes per second, making the process several weeks if waited for completion. Note that data is constantly transmitted, just at a very slow rate, to avoid middleboxes cutting off the connection.

	Version	**Operating System**	**CPU**	**RAM**	**Disk**
Routinator	0.10.1	Ubuntu 18.04 LTS	1× 2.3 GHz	2 GB + 2 GB swap	20 GB
Validator 3	3.2.2021.04.07.12.55	Ubuntu 18.04 LTS	1× 2.3 GHz	2 GB	20 GB
OctoRPKI	1.3.0	Ubuntu 18.04 LTS	1× 2.3 GHz	2 GB + 2 GB swap	20 GB
Fort	1.5.1	Ubuntu 18.04 LTS	1× 2.3 GHz	2 GB	20 GB
RPKI-Prover	0.1.0	Ubuntu 18.04 LTS	1× 2.3 GHz	2 GB + 2 GB swap	20 GB
rpstir2	master-7394f73	Ubuntu 18.04 LTS	1× 2.3 GHz	2 GB + 2 GB swap	20 GB
rpki-client	7.3	OpenBSD 7.0	1× 2.3 GHz	2 GB	20 GB
rcynic	1.0.1544679302	Ubuntu 16.04 LTS	1× 2.3 GHz	2 GB	50 GB

Table 2: The specifications of the machines used for executing the tests.

All tested implementations apart from RPKI-Prover keep waiting forever. RPKI-Prover imposes a maximum transfer duration, and continues with the next repository afterwards.

5.2.6 (F) Broken ROA

Whenever relying party software encounters a ROA, a file with a .roa extension, it expects the structure that belongs to a ROA. However, it is possible to encode any data, and give it a .roa extension. Here the ROA merely consists of an encoded ASCII NUL character. Without proper input validation, this might cause relying party software to malfunction or crash.

Only OctoRPKI and rpstir2 crash when encountering the broken ROA, whereas all other tested implementations show a warning and move on.

5.2.7 (G) Billion laughs attack

The Billion Laughs attack is an XML expansion attack, where an entity “lol1" is defined as “lol", and “lol2" as 10 times “lol1". This is done ten times to create a billion entities [58]. The textual form is small, but XML parsers that try to handle this in memory will likely run out of memory and crash.

Only RPKI-Prover and rpki-client ≤ 7.2 crash when encountering the billion laughs attack. All other tested implementations treat is as a broken repository, and retry it over rsync.

5.2.8 (H) Exponential expansion

Similar to test A, we create a chain of repositories. Instead of having one child, we have 10 children at each level. This means that the amount of CAs and repositories that must be visited becomes $\sum_{i=0}^{d} w^i$, where w is the amount of children and d is the depth. Even at a modest depth of 8, and a width of 10, this already becomes 11,111,111 repositories that must be visited. Even at one repository per second, this will take months, which breaks the expectation that the tree traversal can be done in reasonable time.

All implementations keep retrieving repositories. The order in which the repositories are retrieved does differ. We observed that most use either an depth-first search or breadth-first search approach, and that some will retrieve repositories in concurrent fashion.

5.2.9 (I) Impossible ROA

Much like test F, this is something presented as a ROA that is in fact not a valid ROA. Here we undermine the assumptions made in the structure of the ROA, namely, that an IPv4 address will have a prefix with at most 32 bits and an IPv6 address has a prefix with at most 128 bits. This is similar to the out-of-bounds vulnerability as described in CVE-2021-3761 [38], and we expect a similar impact: either the validator crashes, or the output becomes nonsensical, causing the RPKI to Router (RTR) protocol to terminate.

Only OctoRPKI and rpstir2 crash when encountering the faulty ROA, whereas all other tested implementations show a warning and move on.

5.2.10 (J) ROA ASN overload

Relying party software has two functions: it retrieves data from repositories, and it processes that information so that RPKI-to-Router (RTR) software can relay that information to a router’s BGP Route Origin Validation (ROV) table [10]. The memory of a router is limited. If one were to create ROAs for a prefix for every ASN, that means that suddenly the router has to store an extra 2^{32} entries. Even at a mere 20 bytes for each entry, this yields around 85 GB of data, which is larger than the memory capacity most current routers have.

All tested implementations do not limit the maximum amount of ASNs for a prefix. Some implementations do implement some safeguards, such as a maximum repository size or maximum number of files. However, in all cases we were able to generate a problematic number of entries. These protections can likely be circumvented by, for example, delegating part of the ASNs to children, or using rsync exclusively.

5.2.11 (K) ROA prefix overload

Much like test J, what can be done with ASNs can also be done with prefixes. If an IPv6 /48 has been delegated, then one can create $\sum_{i=0}^{80} 2^i \approx 2^{81}$ unique prefixes for one ASN. Combined with the results from J, one can create $2^{81} \times 2^{32} \approx 2^{133}$ entries. Even at 1 bit per entry, this results in the order of 10^{30} GB of data. The repository does not need to store this amount of data, as the data can deterministically be generated, only the SHA-256 hashes need to be stored at 32 bytes per file.

All tested implementations do not limit the maximum amount of subprefixes for a prefix; rpki-client exits with "excessive runtime (3600 seconds), giving up". That error message comes up after around four to five hours, rather than the expected one hour. We are unsure what the cause of this is.

We are not aware of any router that limits the subprefixes it receives over RTR, nor are we aware of any RTR server software that applies its own filtering or aggregation by default.

5.2.12 (L) Large file

This is a simpler version of test D and G. Instead of using elaborate tactics to trick relying party software into processing a lot of data, a lot of data is actually served. For this test, we define a the URI attribute for the RRDP snapshot as an URL that refers to a file several gigabytes in size, commonly used for bandwidth tests. We expect similar results to D and G, namely memory exhaustion. This file was hosted using an external URI with the random bytes as content.

OctoRPKI, Fort, rpki-client, and rpstir2 crash when encountering the file. Routinator and RPKI-Prover treat is as a broken repository, and retry it over rsync.

5.2.13 (M) XXE on attributes

This test tries an XML eXternal Entities (XXE) attack on an attribute. This should not be possible, as XXE are not allowed in attributes. However, as XXE attacks do happen [31], we wanted to ensure that the used XML parsers also behave according to the specifications. In this test, we try to extract contents of a file on the filesystem, and pass them to our server, by including an XML external entity in the snapshot URI.

All tested implementations do not support it, and neither do the libraries that they use. This is not to say that none of the implementations support XXE, but merely that we have not found a way to exfiltrate data from the host system to the server the repository runs on.

5.2.14 (N) Long paths

Much like D, G, and L, this test tries to exhaust the memory, this time by using a path for the data that is much longer than can be reasonably expected (in the order of megabytes). Our expectation is that natively trying to parse this path will result in a crash due to memory exhaustion, and that trying to write to this path will cause an error from the operating system.

Only OctoRPKI and rpstir2 crash when parsing the long path, whereas all other tested implementations show a warning and move on. We are unsure whether this purely affects the RRDP implementation, or whether this would also be possible over rsync.

5.2.15 (O) Path traversal

This test contains valid data with a path that attempts to write to a folder up from where it should. This is based on an rsync security advisory from December 21st, 2015 [62], where the server could send a file list containing a path with special folders '..' and '.', causing the rsync client to write outside of the destination folder. We attempt to do the same using RRDP, by setting the URI to include '..' in the path. Our expectation is that this potentially allows for remote code execution by being able to write files to arbitrary locations.

Only OctoRPKI, Fort and rpstir2 allow writing outside the intended destination folder. The other tested implementations reject it. We observed that default installation instructions make it easy to accidentally (or even intentionally) run OctoRPKI, Fort and rpstir2 as root. This means that a file like "rsync://example.org/repo/../../etc/cron.daily/evil.roa" could allow for remote code execution on the host machine the relying party software is running on.

6 Discussion

It is important to determine the difficulty to execute this attack, as well as its impact in the short term and long term.

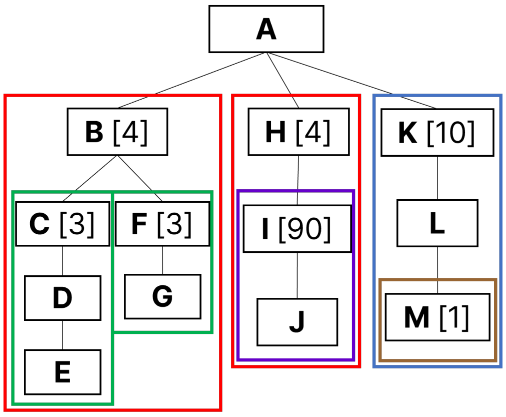
All actively supported implementations had at least one vulnerability that allows them to be disrupted without the need for CA control. This means that an attacker who successfully takes over a repository server, performs a DNS hijack, or has the means to perform a Man-In-The-Middle (MITM) attack, can disrupt the RPKI. As RFC 8182 states that HTTPS certificates may be self-signed [8], this means that any party in the path of any RPKI repository can disrupt RPKI for a large part of the world, without necessarily being part of the RPKI tree. Additionally, controlling part of the RPKI tree is not too difficult. In the case of most RIRs, it consists of a small fee and an identity check [61].

Parents, such as RIRs and NIRs, can monitor this, and revoke a certificate when malicious behaviour is discovered. The main issue is that there is no guarantee that a repository serves the same content to all parties. A malicious party can purposefully exclude the parent from their attack, or direct their attack at only one victim based on the IP address or ASN. An attacker may also only deploy this attack once. This slows down the process of getting the certificate of the malicious party revoked, especially if the malicious party also has non-malicious objects, such as a hijacked legitimate publication point. This is not merely a technical issue, but also a political one.

Once the threat has been removed from the tree, the operators of relying party software often also need to manually restart their software. This requires adequate monitoring. From a previous bug in Fort, where the software would crash when encountering a BGPsec certificate [52], we observed that roughly 3 out of 10 instances did not come back online within a month. We expect that the long term effects of our attacks will be similar, meaning that 30% of relying party instances would not come back online for months after a successful attack, even if that attack only lasted mere hours.

Most of these vulnerabilities have now been resolved [39–47, 50]. However, as of 1 January 2022, insecure versions of relying party software is still present in the default repositories. Ongoing data provided by Kristoff et al. [25] shows that the now unsupported RIPE NCC RPKI Validator 3 is still used by approximately 5% of users. Debian 11 still has rpki-client 6.8, OctoRPKI 1.2.2, and Fort 1.5.0 in its default software repository – we found no evidence that the security fixes have been backported. Ubuntu 20.04 still has Fort 1.2.0 in its default software repository. Ubuntu 21.04 and 21.10 also provide rpki-client 6.8 and OctoRPKI 1.2.2. The Fort versions for Ubuntu 21.04 and 21.10 are newer (1.5.0 and 1.5.1 respectively), but are also both still vulnerable. rpstir2 and rcynic are also both still vulnerable with no known fix as of 1 January 2022.

Additionally, we want to draw the attention to the fact that all implementations are still vulnerable to test H. This is a problem that is thus-far unsolved, as in discussions with relying party software developers we came to the conclusion that there are to our knowledge no heuristics with which malicious parties can be adequately detected without causing collateral damage to non-malicious parties. When all information of the tree is required, and the nodes in the tree are untrusted, then limits to the structure are required for the tree to be evaluable within a set time limit. The problem is that without outside information, it is not possible to tell which node belongs to the malicious set, and which one is not. Once a parent create a child, and hands the key to a third-party, that third-party has exactly the same ability under its subtree as the parent has. Whilst for test A a depth limit is an adequate solution, exponentially increasing repositories already become problematic even at a very shallow depth. Some RIRs and NIRs may have in the order of 10,000 CAs, which makes width restrictions difficult, as our malicious node may be at the same level as NIRs, which may have genuine reason for their structure. This problem is still present to this day. Setting limits that fit the current shape of the trees, whilst also preventing malicious nodes from being able to disrupt the RPKI, is to our knowledge not possible. This means that either the structure of the RPKI tree must change, certain aspects must be hard-coded in the relying party software, or the parent must be able to provide the required outside information about what can be expected from the child. A draft to add the latter has been submitted [68], but is still being discussed. It allows a parent to provide "hints" about what limits should apply to their children as shown in figure 7.



Node	Limits	Eff. limit
A	∞	∞
B, H	∞, 4	4
C, D, E, F, G	∞, 4, 3	3
I, J	∞, 4, 90	4
K, L	∞, 10	10
M	∞, 10, 1	1

Figure 7: An example of how RPKI tree hints could work. Shown is an example tree, with the maximum descendants allowed for each node indicated as number in the brackets. The limit is applied recursively for all descendants, and the effective limit is the strictest of all parents. Since B has an effective limit of 4, and C to G are 5 nodes, Relying parties may decide to not retrieve either E or G, depending on the traversal order. What traversal order to use is not specified.

7 Ethical considerations

As a realistic possible outcome of our research was real vulnerabilities, we decided to test in a manner that would allow us to verify the exploit, without disrupting actual day-to-day RPKI operations. Primarily, we decided not to test this on the "production" RPKI, but rather create our own separate tree. As our research resulted in real exploitable security issues, we started a multi-party coordinated vulnerability disclosure (CVD) process with the National Cyber Security Centre of the Netherlands (NCSC-NL), where the NCSC-NL coordinated the CVD. The NCSC-NL decided to contact implementers of

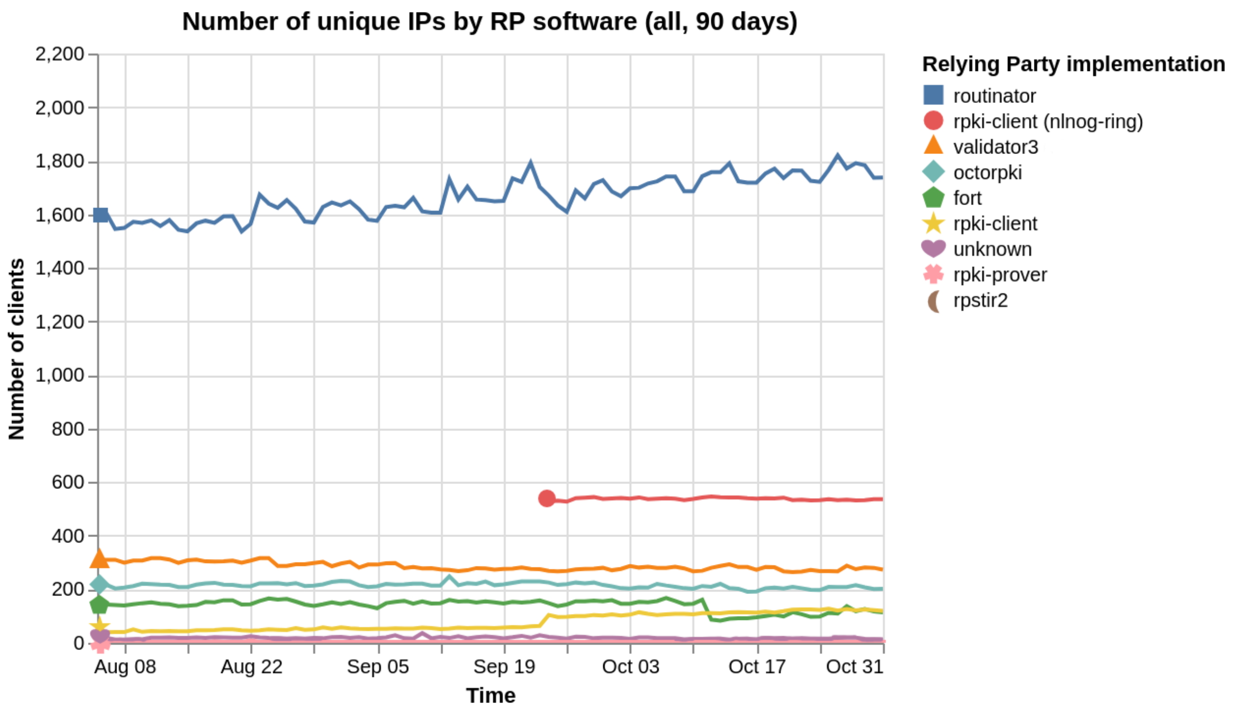


Figure 8: Absolute relying party usage as observed by the RIPE NCC in the months leading up to the disclosure date. All dates are in 2021. rpki-client was split into two, as part of their automated integration testing is done using the NLNOG ring, a cluster of servers. The graph was provided as a courtesy by the RIPE NCC.

actively maintained relying party software. The decision who to contact was based on whether the parties had 1. a clear vulnerability disclosure process compatible with the principle of confidential coordinated vulnerability disclosure [49] 2. clear points of contact to report security issues. At the time of starting the CVD process, the parties included were Routinator by NLnet Labs, RPKI Validator 3 by the RIPE NCC, OctoRPKI by Cloudflare, and Fort Validator by NIC México. There was also an explicit decision by the NCSC-NL not to contact the developers of rpki-client, from the OpenBSD project, due to their publicly announced policy of full disclosure of vulnerabilities [55] and due to earlier issues with disclosures like KRACK [69]. The secondary consideration was ensuring that the users of the various relying party software would not be harmed. We used data provided by the RIPE NCC, shown in figure 8, to determine the most widely used relying party software. We did not notice any significant change in these figures over the course of the CVD. Other projects that were not contacted at the start of the CVD process were:

rcynic Developed by Dragon Research Labs, due to the fact that there is no security policy or contact, the software sees little to no active use and the software had not been actively maintained since 2016 as of the time the CVD process started;

rpstir2 Developed by ZDNS, due to the fact that there is no security policy or contact and the software did not build or work at the time the CVD process started;

RPKI-Prover Developed by an individual developer, due to the fact that there is no security policy or contact, and the software saw little or no active production use at the start of the CVD process.

The full timeline of the CVD can be found in appendix A. We have contacted all parties involved before publication.

8 Conclusion

We have looked into the unique characteristics of the RPKI, where unlike other common protocols all information must be retrieved first, and used that to create a threat model where an attacker that wanted to disrupt RPKI operations had full control over a certificate authority and its RRDP publication point. We have developed an exploit framework based on that threat model, and tested how current RPKI relying party software deals with these threats. We showed that in many cases the case where the certificate authority or publication point was malicious was inadequately considered by relying party software, allowing a publication point to disrupt the entirety of RPKI. Additionally, we showed that the protocol design seems in its current state incapable of technically ensuring all the necessary assumptions about the protocol to function properly and securely to hold, making it impossible for relying party software developers to adequately prevent disruption by a malicious certificate authority without collateral damage. Lastly, we have described the steps and considerations for reporting these issues to all parties involved.

Future work

So far we have only looked into the RRDP threat model from a relying party perspective. We believe other aspects of the RPKI would benefit from similar scrutiny.

Acknowledgements

We would like to thank, in no particular order, the people at: NLnet Labs, RIPE NCC, the National Cyber Security Centre of the Netherlands (NCSC-NL), NIC México, Cloudflare, and LACNIC for their support during the vulnerability disclosure process.

Availability

The code for the test framework, as well as the manual to set it up, is available under the Affero General Public License version 3 on https://gitlab.com/Koenvh/relying-party-resiliency-platform.

References

[1] Donald E. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, January 2011. URL: https://rfc-editor.org/rfc/rfc6066.txt, doi:10.17487/RFC6066.

[2] Derek Atkins and Rob Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, August 2004. URL: https://rfc-editor.org/rfc/rfc3833.txt, doi:10.17487/RFC3833.

[3] Alexander Azimov, Eugene Bogomazov, Randy Bush, Keyur Patel, and Job Snijders. Verification of AS_PATH Using the Resource Certificate Public Key Infrastructure and Autonomous System Provider Authorization. Internet-Draft draft-ietf-sidrops-aspa-verification-07, Internet Engineering Task Force, February 2021. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-sidrops-aspa-verification-07.

[4] Jason Bau and John C Mitchell. A security evaluation of dnssec with nsec3. In *NDSS*, page 18, 2010.

[5] Mike Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-34, Internet Engineering Task Force, February 2021. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34.

[6] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and Dave Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008. URL: https://rfc-editor.org/rfc/rfc5280.txt, doi:10.17487/RFC5280.

[7] Tim Bruijnzeels, Randy Bush, and George G. Michaelson. Resource Public Key Infrastructure (RPKI) Repository Requirements. Internet-Draft draft-ietf-sidrops-prefer-rrdp-01, Internet Engineering Task Force, October 2021. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-ietf-sidrops-prefer-rrdp-01.

[8] Tim Bruijnzeels, Oleg Muravskiy, Bryan Weber, and Rob Austein. The RPKI Repository Delta Protocol (RRDP). RFC 8182, July 2017. URL: https://rfc-editor.org/rfc/rfc8182.txt, doi:10.17487/RFC8182.

[9] Randy Bush. The Resource Public Key Infrastructure (RPKI) Ghostbusters Record. RFC 6493, February 2012. URL: https://rfc-editor.org/rfc/rfc6493.txt, doi:10.17487/RFC6493.

[10] Randy Bush and Rob Austein. The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1. RFC 8210, September 2017. URL: https://rfc-editor.org/rfc/rfc8210.txt, doi:10.17487/RFC8210.

[11] Margaux Canet, Amrit Kumar, Cédric Lauradoux, Mary-Andréa Rakotomanga, and Reihaneh Safavi-Naini. Decompression quines and anti-viruses. *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, Mar 2017. doi:10.1145/3029806.3029818.

[12] Cloudflare. Slowloris ddos attack, 2021. URL: https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/.

[13] Danny Cooper, Ethan Heilman, Kyle Brogle, Leonid Reyzin, and Sharon Goldberg. On the risk of misbehaving rpki authorities. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, HotNets-XII, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2535771.2535787.

[14] Borislav Djordjevic and Valentina Timcenko. Ext4 file system performance analysis in linux environment. In *Proceedings of the 11th WSEAS International Conference on Applied Informatics and Communications, and Proceedings of the 4th WSEAS International Conference on Biomedical Electronics and Biomedical Informatics, and Proceedings of the International Conference on Computational Engineering in Systems Applications*, AIASABEBI'11, page 288–293, Stevens Point, Wisconsin, USA, 2011. World Scientific and Engineering Academy and Society (WSEAS).

[15] Benjamin Dowling, Marc Fischlin, Felix Günther, and Douglas Stebila. A cryptographic analysis of the tls 1.3 handshake protocol. *Journal of Cryptology*, 34(4):1–69, 2021.

[16] Roy T. Fielding and Mark Nottingham. Additional HTTP Status Codes. RFC 6585, April 2012. URL: https://rfc-editor.org/rfc/rfc6585.txt, doi:10.17487/RFC6585.

[17] Yossi Gilad, Omar Sagga, and Sharon Goldberg. Maxlength considered harmful to the rpki. *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, Oct 2017. doi:10.1145/3143361.3143363.

[18] Sharon Goldberg. Why is it taking so long to secure internet routing? routing security incidents can still slip past deployed security defenses. *Queue*, 12(8):20–33, aug 2014. doi:10.1145/2668152.2668966.

[19] Dan Goodin. Suspicious event hijacks amazon traffic for 2 hours, steals cryptocurrency, Apr 2018. URL: https://arstechnica.com/information-technology/2018/04/suspicious-event-hijacks-amazon-traffic-for-2-hours-steals-cryptocurrency/.

[20] Russ Housley. Cryptographic Message Syntax (CMS). RFC 5652, September 2009. URL: https://rfc-editor.org/rfc/rfc5652.txt, doi:10.17487/RFC5652.

[21] Geoff Huston, Robert Loomans, and George G. Michaelson. A Profile for Resource Certificate Repository Structure. RFC 6481, February 2012. URL: https://rfc-editor.org/rfc/rfc6481.txt, doi:10.17487/RFC6481.

[22] Geoff Huston and George G. Michaelson. Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs). RFC 6483, February 2012. URL: https://rfc-editor.org/rfc/rfc6483.txt, doi:10.17487/RFC6483.

[23] Stephen Kent and Andrew Chi. Threat Model for BGP Path Security. RFC 7132, February 2014. URL: https://rfc-editor.org/rfc/rfc7132.txt, doi:10.17487/RFC7132.

[24] Stephen Kent, Geoff Huston, and George G. Michaelson. Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI). RFC 6489, February 2012. URL: https://rfc-editor.org/rfc/rfc6489.txt, doi:10.17487/RFC6489.

[25] John Kristoff, Randy Bush, Chris Kanich, George Michaelson, Amreesh Phokeer, Thomas C. Schmidt, and Matthias Wählisch. On measuring rpki relying parties. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 484–491, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3419394.3423622.

[26] Paul J. Leach, Rich Salz, and Michael H. Mealling. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122, July 2005. URL: https://rfc-editor.org/rfc/rfc4122.txt, doi:10.17487/RFC4122.

[27] Matt Lepinski and Stephen Kent. An Infrastructure to Support Secure Internet Routing. RFC 6480, February 2012. URL: https://rfc-editor.org/rfc/rfc6480.txt, doi:10.17487/RFC6480.

[28] Matt Lepinski, Stephen Kent, Geoff Huston, and Rob Austein. Manifests for the Resource Public Key Infrastructure (RPKI). RFC 6486, February 2012. URL: https://rfc-editor.org/rfc/rfc6486.txt, doi:10.17487/RFC6486.

[29] Matt Lepinski, Derrick Kong, and Stephen Kent. A Profile for Route Origin Authorizations (ROAs). RFC 6482, February 2012. URL: https://rfc-editor.org/rfc/rfc6482.txt, doi:10.17487/RFC6482.

[30] Matt Lepinski and Kotikalapudi Sriram. BGPsec Protocol Specification. RFC 8205, September 2017. URL: https://rfc-editor.org/rfc/rfc8205.txt, doi:10.17487/RFC8205.

[31] Mitre. CVE-2015-3451, 2015. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3451.

[32] Mitre. CVE-2016-2108, 2016. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2108.

[33] Mitre. CVE-2016-6308, 2016. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6308.

[34] Mitre. CVE-2018-1000613, 2018. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1000613.

[35] Mitre. CVE-2019-16159, 2019. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-16159.

[36] Mitre. CVE-2019-17359, 2019. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-17359.

[37] Mitre. CVE-2019-6659, 2019. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-6659.

[38] Mitre. CVE-2021-3761, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3761.

[39] Mitre. CVE-2021-3907, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3907.

[40] Mitre. CVE-2021-3908, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3908.

[41] Mitre. CVE-2021-3909, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3909.

[42] Mitre. CVE-2021-3910, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3910.

[43] Mitre. CVE-2021-3911, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3911.

[44] Mitre. CVE-2021-3912, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3912.

[45] Mitre. CVE-2021-43172, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43172.

[46] Mitre. CVE-2021-43173, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43173.

[47] Mitre. CVE-2021-43174, 2021. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43174.

[48] P. Mockapetris. Domain names - concepts and facilities. RFC 1034, November 1987. URL: https://rfc-editor.org/rfc/rfc1034.txt, doi:10.17487/RFC1034.

[49] Nationaal Cyber Security Centrum. Leidraad coordinated vulnerability disclosure, Jul 2019. URL: https://www.ncsc.nl/documenten/publicaties/2019/mei/01/cvd-leidraad.

[50] Nationaal Cyber Security Centrum. NCSC-2021-0987, 2021. URL: https://www.ncsc.nl/actueel/advisory?id=NCSC-2021-0987.

[51] Nationaal Cyber Security Centrum. Upcoming announcement of rpki cvd procedure, Oct 2021. URL: https://web.archive.org/web/20211029165109/https://english.ncsc.nl/latest/news/2021/october/29/upcoming-announcement-of-rpki-cvd-procedure.

[52] NICMx/FORT-validator. 1.5.3 crashing repeatedly starting about a hour ago, 2021. URL: https://github.com/NICMx/FORT-validator/issues/65.

[53] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, May 1996. URL: https://rfc-editor.org/rfc/rfc1945.txt, doi:10.17487/RFC1945.

[54] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. URL: https://rfc-editor.org/rfc/rfc2616.txt, doi:10.17487/RFC2616.

[55] OpenBSD. Openbsd security, 2021. URL: https://www.openbsd.org/security.html.

[56] OpenBSD. Openrsync, 2021. URL: https://www.openrsync.org/.

[57] OWASP. Rest security cheat sheet, 2021. URL: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html.

[58] OWASP. Xml security cheat sheet, 2021. URL: https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html.

[59] Yakov Rekhter, Susan Hares, and Tony Li. A Border Gateway Protocol 4 (BGP-4). RFC 4271, January 2006. URL: https://rfc-editor.org/rfc/rfc4271.txt, doi:10.17487/RFC4271.

[60] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018. URL: https://rfc-editor.org/rfc/rfc8446.txt, doi:10.17487/RFC8446.

[61] RIPE NCC. Become a ripe ncc member, 2021. URL: https://www.ripe.net/participate/member-support/become-a-member.

[62] Rsync. Rsync security advisories, 2021. URL: https://rsync.samba.org/security.html.

[63] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525, May 2015. URL: https://rfc-editor.org/rfc/rfc7525.txt, doi:10.17487/RFC7525.

[64] Kris Shrishak and Haya Shulman. Limiting the power of rpki authorities. In *Proceedings of the Applied Networking Research Workshop*, ANRW '20, page 12–18, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3404868.3406674.

[65] Tom Strickx. How verizon and a bgp optimizer knocked large parts of the internet offline today, Jun 2019. URL: https://blog.cloudflare.com/how-verizon-and-a-bgp-optimizer-knocked-large-parts-of-the-internet-offline-today/.

[66] Nathalie Trenaman. Update on ripe roadmap and experiences, 2021. RIPE 82. URL: https://ripe82.ripe.net/archives/video/602/.

[67] Koen van Hove. Improving the resiliency of rpki relying party software, Nov 2021. URL: https://labs.ripe.net/author/koen-van-hove/improving-the-resiliency-of-rpki-relying-party-software/.

[68] Koen van Hove. Tree Hints for the Resource Public Key Infrastructure (RPKI). Internet-Draft draft-kwvanhove-sidrops-rpki-tree-hints-01, Internet Engineering Task Force, December 2021. Work in Progress. URL: https://datatracker.ietf.org/doc/html/draft-kwvanhove-sidrops-rpki-tree-hints-01.

[69] Mathy Vanhoef. Key reinstallation attacks - breaking wpa2 by forcing nonce reuse, 2017. URL: https://www.krackattacks.com/.

[70] Matthias Wählisch, Robert Schmidt, Thomas C. Schmidt, Olaf Maennel, Steve Uhlig, and Gareth Tyson. Ripki: The tragic story of rpki deployment in the web ecosystem. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2834050.2834102.

[71] Zhiwei Yan and Jong-Hyouk Lee. Bgpchain: Constructing a secure, smart, and agile routing infrastructure based on blockchain. *ICT Express*, 2021. URL: https://www.sciencedirect.com/science/article/pii/S2405959520304938, doi:https://doi.org/10.1016/j.icte.2020.12.005.

A Vulnerability disclosure timeline

Below is the full timeline from the start of the project to the end of the CVD process. All dates are in 2021.

May	Start of the research project
June	First set of vulnerabilities discovered, decision to start a CVD process made, NCSC-NL is contacted by the researchers
July	Discussion between the researchers and NCSC-NL on the CVD process to follow
August	Identification of stakeholders in the CVD process to contact
12 August	Start CVD process – NCSC-NL contacts relying party software implementers to notify them that vulnerabilities have been found and asks them to confirm willingness to participate under embargo on providing patches with a coordinated release tentatively scheduled for November 8, 2021
12 August	RIPE NCC responds that they do no longer support Validator 3, and that they will not update it
17 August	The implementers have all responded to the notification and have received information on the vulnerabilities
25 October	All three participating implementations (Routinator, OctoRPKI and Fort) have patches ready and releases on standby
25 October	Due to early availability of fixes, publication date is moved forward to November 1, 2021, with notification of all other parties (rpki-client, rcynic, rpstir2, and RPKI-Prover) set for October 27, 2021
26 October 18h	Pull request with fixes to OctoRPKI is inadvertently made publicly visible. Automated GitHub notifications are sent to all users following this project, including parties not notified yet in the CVD process
26 October 20h	OctoRPKI pull request is removed from the public record
26 October 21h	Decision is made to move notification of other parties forward by a few hours in light of the public PR
26 October 21h	Notifications are sent to rpki-client, rcynic, rpstir2, and RPKI-Prover. These notifications are sent by the researcher instead of NCSC-NL, in deviation from the process, in the interest of speed in notification
27 October	OpenBSD responds to notification of rpki-client, indicating they do not agree to the terms because the time between notification and publication is too short, and not wanting to agree to an embargo a priori
27 October	NCSC-NL gives OpenBSD the option to negotiate an entirely new deadline provided OpenBSD agrees to keep the disclosed information confidential under embargo
27 October	OpenBSD responds negatively to NCSC-NL and indicates they are not willing to agree to an embargo and request not to be contacted again in the remainder of the CVD process
27 October	Email to one of the rcynic developers has bounced, and another contact attempt with a corrected email address is made to both developers
27 October	OpenBSD developers publicly criticise the CVD process in multiple online forums
29 October	rcynic point of contact publicly denounces CVD process on NANOG mailing list including an unredacted copy of the notification mail in their post
29 October	NCSC-NL privately notifies the rcynic point of contact about the omission in the notification message describing the rest of the process and timeline
29 October	NCSC-NL issues a public statement about the CVD process due to ongoing discussion in public forums [51]
30 October	OpenBSD changes its position and agrees to keep the disclosed information confidential under embargo – is informed of the vulnerabilities
31 October	Disclosure date is moved to 9 November
9 November 15h	Embargo is lifted, updates released, CVD is over