rpkiller: Threat Analysis of the BGP Resource Public Key Infrastructure

KOEN VAN HOVE, Science Park, Amsterdam, The Netherlands & University of Twente, The Netherlands
JEROEN VAN DER HAM-DE VOS and ROLAND VAN RIJSWIJK-DEIJ, University of Twente, The Netherlands

The Resource Public Key Infrastucture (RPKI) has been created to solve security shortcomings of the Border Gateway Protocol (BGP). This creates an infrastructure where resource holders (autonomous systems) can make attestations about their resources (IP-subnets). RPKI Certificate Authorities make these attestations available at Publication Points. Relying Party software retrieves and processes the RPKI-related data from all publication points, validates the data and makes it available to routers so they can make secure routing decisions. We contribute to this work by doing a threat analysis for Relying Party software, where an attacker controls a Certificate Authority and Publication Point. We implement a prototype testbed to analyse how current Relying Party software implementations react to scenarios originating from that threat model.

Our results show that all current Relying Party software was susceptible to at least one of the identified threats. In addition to this, we also identified threats stemming from choices made in the protocol itself. Taken together, these threats potentially allowed an attacker to fully disrupt all RPKI Relying Party software on a global scale. We elaborate on our process, and we discuss the types of responses we received from other parties. We performed a Coordinated Vulnerability Disclosure to the implementers.

CCS Concepts: • **Networks** → **Network protocol design**; *Network security*; *Security protocols;*

Additional Key Words and Phrases: Routing security, RPKI, responsible disclosure

ACM Reference format:
Koen van Hove, Jeroen van der Ham-de Vos, and Roland van Rijswijk-Deij. 2023. rpkiller: Threat Analysis of the BGP Resource Public Key Infrastructure. *Digit. Threat. Res. Pract.* 4, 4, Article 58 (October 2023), 24 pages.
https://doi.org/10.1145/3617182

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 a residential 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), advertises (1) who they are; (2) what their IP subnet is; and (3) who can be reached (indirectly) via them. They send this to the peers they are connected to, who then further distribute this information throughout the network.

Authors' addresses: K. van Hove, NLnet Labs & University of Twente, Science Park 400 1098 XH, Amsterdam, The Netherlands; e-mail: koen@nlnetlabs.nl; J. van der Ham-de Vos and R. van Rijswijk-Deij, University of Twente, Postbus 217, 7500 AE Enschede, The Netherlands; e-mails: {j.vanderham, r.m.vanrijswijk}@utwente.nl.

2576-5337/2023/10-ART58
https://doi.org/10.1145/3617182

Eventually, routers end up with an overview of how to reach different prefixes through a path through the AS network [62].

To overcome the security challenges in inter-domain routing, the Resource Public Key Infrastructure (RPKI) was proposed in 2011. RPKI is a public key infrastructure framework that aims to work together with BGP [31]. RPKI aims 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.

RPKI currently solves the verification issue by providing attestation using certificates. Similar to the WebPKI, there are five root authorities that form the trust anchors of the RPKI system. These are 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 on their own publication point, but RPKI can also be delegated, where customers run their own certificate authority (CA) and host their own publication point, which is then linked to from the delegating party.

Validation happens through Relying Party (RP) software, which retrieves ROAs from the publication points recursively, validates the data, and then makes the results available to a router so it can reject invalid route announcements. Deployment of RPKI is increasing rapidly [11] and currently covers 42% of all prefix announcements. In the context of the latter, the question that arises is whether a dishonest delegatee, i.e., one that has the intent to sabotage the system, can abuse their power to disrupt Relying Party software, and potentially RPKI as a whole.

So far most have looked into what security the RPKI provides, such as Wählisch et al. [75]. These all presume that the RPKI is working properly. What to our knowledge has not been analysed is whether this presumption holds even when a malicious actor on the publication side is introduced. For that reason, in this article we ask ourselves: *How can a dishonest Publication Point or Certificate Authority disrupt the operations of Relying Party software?* To answer this question, we:

(1) Develop a threat model for RPKI focussing on the abilities of a dishonest publication point and/or certificate authority;
(2) Create a proof-of-concept exploit framework and an overview of the behaviour of relying party software with regards to the threats we identified;
(3) Examine the efforts we took to resolve these threats, and share our lessons learned.

Outline – The remainder of this article is organised as follows. In Section 2, we provide a brief background on BGP and the RPKI. In Section 3, we discuss related work. In Section 4, we introduce our threat model. In Section 5, we introduce practical test cases based on the threat model and test those against eight open-source Relying Party implementations. In Section 6, we reflect on the results of these tests. In Section 7, we discuss ethical considerations and describe the process we used to notify RP implementers of the vulnerabilities we identified. In Section 8, we describe our efforts to resolve these issues and the responses it created. In Section 9, we present our conclusions and discuss future work.

2 BACKGROUND

2.1 BGP

The Border Gateway Protocol (BGP) is a protocol that aims to exchange routing and reachability information for Autonomous Systems (AS). BGP generally comes in two varieties: internal (within an AS) and external (between ASs). Historically, when talking about BGP, the exterior variant is meant, which is the variant that has been extended with RPKI. For the purpose of this article, interior BGP is out of scope.

Let us explain how BGP works by means of example, as shown in Figure 1. The nodes `64501` to `64505` represent our ASs, and the lines represent connections and BGP sessions between them. Initially, the nodes do not have any

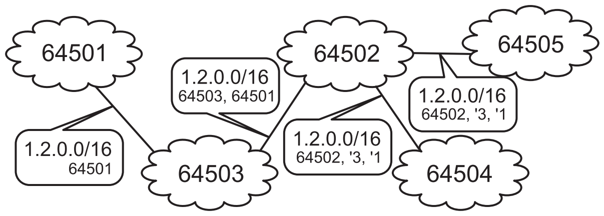


Fig. 1. An example of a network with five autonomous systems, `64501` to `64505`, with lines representing interconnections. An example of the BGP advertisement for `1.2.0.0/16` from `64501` is shown.

routing information. Then, AS `64501` *advertises* to `64503` that `64501` is the destination for `1.2.0.0/16`, giving an *AS path* of ⟨`64501`⟩. AS `64503` learns that information and advertises to `64502` that `1.2.0.0/16` can be reached via ⟨`64503, 64501`⟩, that is, traffic to `1.2.0.0/16` should be sent to `64501` through `64503`. Repeat this, and `64504` and `64505` learn that a path to `1.2.0.0/16` is ⟨`64502, 64503, 64501`⟩. This is depicted in Figure 1.

If `64505` would also advertise `1.2.0.0/16`, then `64504` will learn that ⟨`64502, 64505`⟩ is also a route for that prefix and thus a second route exists. The BGP decision process at 64502 will then select as best route one of the two (generally the one with the shortest path [62]) and will propagate the new decision to 64504 — and potentially 64503, depending on the nature of the economic relationship between 64503 and 64504 [19] — with path (64502, 64505). For the very same rationale on AS path length, the BGP decision process at 64503 will prefer the path via 64501 rather than via 64502. However, this would still mean that traffic from 64502 and 64504 to 1.2.0.0/16 would be routed. This is an example of a routing hijack.

The main issue facing BGP currently is that it is largely based on good faith. By default, anyone can send any BGP announcement, with one's own AS Number as the destination for *any* prefix [21]. This means that someone other 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 [22, 69]. BGP forms an integral part of the worldwide inter-AS routing system and is not easily replaced. While a secure version of the BGP protocol exists (called BGPsec [34]), this protocol has gained little traction and is not actively used by operators due to deployment challenges [11, 21].

In the scenario above, only part of the network would be affected by the routing hijack due to the AS-path length preference. Another way that routes can be hijacked is by advertising a more specific (i.e., smaller) subnet. 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`.

Real-life examples of (accidental) route hijacks include redirecting all traffic aimed for YouTube to Pakistan [72], popular destinations being rerouted via Russia [70], and traffic aimed for Amazon Route 53 being redirected for two hours [22].

Summarizing, BGP does not provide any means to verify any of the claims made by an AS. 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. In practice, the situation is more nuanced and every AS has its own policy for routing, for example, regarding the maximum prefix length. There are also many more ways in which routing hijacks can occur [10], but these are out of the scope of this article.

2.2 Increasing BGP Security

RPKI was introduced in 2011 to attempt to solve the attestation problem for IP prefix announcement. There have been earlier attempts at making BGP more secure, such as BGPsec [34], but the main hindrance has been that they required a fundamental change to the BGP protocol and (near) universal deployment, thus requiring a

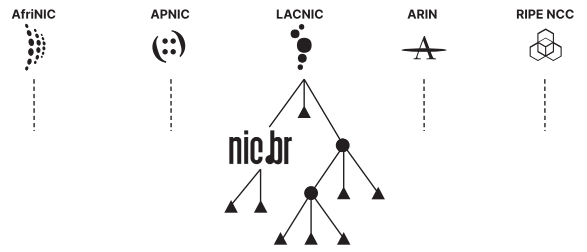


Fig. 2. Example of a tree showing the trust hierarchy in RPKI. The example tree originates from LACNIC. Every circle as well as the RIRs and the National Internet Registry (NIR) of Brazil (.br), represent a CA. Every triangle represents an ROA. Every subordinate CA can delegate (part of) their resources to further child CAs. The five trees of the five RIRs are in principle independent.

flag day. This makes BGPsec difficult to implement. RPKI is a separate infrastructure that can be implemented gradually, while already bringing security benefits. RPKI and BGPsec can work in conjunction with another but are not dependent on one another. RPKI aims to enable creating attestations about IP prefixes, but by advertising Route Origin Authorizations ROAs [33]. A ROA contains an ASN and one or more IP prefixes, combined with a cryptographic signature. Upon receipt of an announcement from a peer, a BGP speaker can look up whether there exists a ROA that covers this prefix. The result can be: (1) valid — a ROA exists, and the origin ASN matches the ROA; (2) invalid — a ROA exists, and the origin ASN does not match the ROA or the prefix length does not match the length specified in the ROA; (3) unknown (or "not found") — there is no ROA for this IP prefix [25]. It is then up to the router to determine how to proceed, and can solve some of the authentication issues inherent in BGP.

It is important to note that a ROA only aims to protect the final destination for a BGP path (called the "origin"), meaning that a valid destination does not guarantee an honest path [21]. Work is on the way to enable full path attestation through the RPKI with, for example, ASPA [2]. 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.

To validate the signature of a ROA, a standard PKI setup using X.509 certificates [4] is used. There are five generally agreed upon trust anchors in the RPKI, namely, the five Regional Internet Registries (RIRs): AfriNIC, APNIC, ARIN, LACNIC, and RIPE [31]. The RIRs can provide services for their customers directly or delegate to subordinate certificate authorities, such as National Internet Registries (NIRs), or to organisations that prefer to manage their own ROAs. These subordinate certificate authorities can again create their own subordinate certificate authorities, and so on.

The RIRs also provide the lookup service by hosting a repository publication point (PP), which is the place where the ROAs, certificates, and other objects can be downloaded from. These publication points are referenced in the certificates [24]. 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 2.

In practice, RPKI is deployed as follows. 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 specific to the RPKI based on HTTPS and XML, contain signed objects. Signed objects are currently mainly ROAs, although Ghostbusters records [7] have been added as well. 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 repository, 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 [24]. An example of a repository can be seen in Figure 3.

Recently, more subordinate certificate authorities and repositories have appeared [29]. Relying party software, the software that collects all the ROAs and creates the lookup table, traverses these subordinate certificate

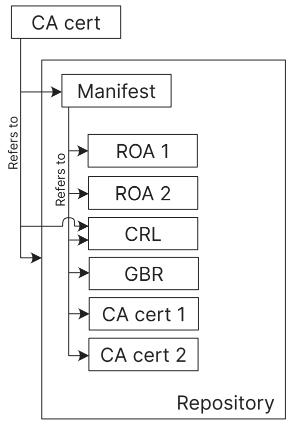


Fig. 3. An example of a repository in RPKI. The CA cert contains an extension that states where the repository can be found and what file inside the repository the manifest (MFT) is. The manifest then contains entries to the ROAs, Certificate Revocation List (CRL), Ghostbusters Record (GBR), and child CA certs. The objects in the repository (MFT, ROA, CRL, GBR, and CA certs) are CMS signed objects [23] using a certificate signed by the CA.

authorities and their repositories. Previously, all repositories belonged to trusted parties, such as RIRs, NIRs, or a select few miscellaneous parties that were generally well trusted. With the growth in delegated RPKI, this implicit faith in repository operators is no longer a given.

2.3 CA and Repository

For the purpose of this work, it is important to differentiate between the notion of a repository and a CA:

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

Repository: A repository 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. This repository is published at a publication point.

In most cases, the repository operator also operates the certificate authority and publication point, but these can be two different entities. For example, an NIR can operate the repository publication point, whilst the actual signing is done by their customers directly. Additionally, in the case in which 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 RPKI Communication Protocols

The RPKI supports two protocols for communication with publication points: rsync and RRDP. Initially, only rsync was used [31], and support for rsync is, at the moment of writing, still required. All common relying party software, however, now also supports the RPKI Repository Delta Protocol (RRDP), as do all of the repository publication points. There are several known issues associated with the use of rsync that have spurred the adoption of RRDP. These issues are not merely theoretical, but have been confirmed in practice [5]. We list important ones below.

(1) *There is no formal specification of the rsync protocol.* Instead, rsync is specified by its implementation. All current relying party implementations 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 [59]. The use of rsync also adds a layer of difficulty for relying party software, as they need to interact with an external rsync process. RRDP, on the other hand, is a defined standard [6].

(2) *Repositories that use rsync are plagued by non-atomic updates* [6, 71], i.e., if the content of a repository changes whilst a relying party is retrieving data, that attempt will fail as the manifest does not match the retrieved data. RRDP does not have this issue as the files are available on a URL that is unique to the session ID and serial number, such that the content is always the same.

(3) *It is trivial to execute a successful denial-of-service attack on an 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 all of these. This causes the rsync daemon to be overloaded and blocks honest parties from retrieving the data from the repository over the rsync protocol. More sophisticated requests, such as specially prepared `--excludes`, allow a malicious party to let the rsync daemon use an arbitrary amount of memory. This violates the RPKI standard, RFC 6481, which states that "The publication repository SHOULD be hosted on a highly available service and high-capacity publication platform" [24]. Due to the reference implementation nature of rsync, we consider resolving this issue infeasible. As RRDP are essentially static files served over HTTPS, this problem does not exist there [6].

(4) *Disrupting the operation of rsync clients is easy*, 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, as all limits in rsync, such as `--max-size` and `--exclude`, are merely kind requests to the rsync daemon not to include these files, and the rsync client does not actually check whether this is the case. RRDP does not support these filters.

Given these issues in rsync, as well as the current uptake of RRDP, we expect rsync to be fully replaced in practice by RRDP. As a consequence, the main focus of this work will be RRDP and we only consider rsync if the issue transcends the communication protocol used.

3 RELATED WORK

Threat models for IETF protocols – the Internet Engineering Task Force (IETF) is the main standards development organisation for the Internet [26]. These voluntary standards are made by the community for the community. RRDP, as mentioned in the previous section, are such a standard, but the protocol for rsync is not. The technical documents produced there are called RFCs (based on their historical title "Request for Comments"). While RFCs typically include a section on security considerations, it is less common for IETF protocols to be analysed using (semi-)formal threat models.

As discussed in the previous section, for the BGP itself, a secure variant has also been specified by the IETF. Its specifications discuss potential threats to integrity and verifiability of data, but lacks a discussion of threats to availability [34]. In contrast, in this work, we specifically focus on availability threats to the RPKI.

An example that stands out is TLS 1.3, which unlike previous versions of the Transport Layer Security protocol and its predecessor, SSL, does have well-described security models, and has undergone extensive formal verification [16, 63]. TLS 1.3 is also standardised at the IETF. We note that these analyses also focus mainly on confidentiality, authenticity, and integrity, but not on availability. Isolated cases of availability do, however, exist (e.g., [40]). An interesting example of what vulnerabilities can be uncovered in earlier versions of the TLS protocol — that lack a well-described security model — is given by De Ruiter and Poll [14]. Their work, which relies on automated protocol fuzzing, is even able to identify a particular version of TLS implementations.

Studies of the RPKI – 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. [75] look at the deployment of RPKI in the context of web hosting and which attacker models RPKI prevents. There has been research into the security of RPKI itself, such as the importance of consistency of objects [32], the lifetime of objects and certificates [28], and the use of the maxLength attribute [20]. Work by Chung et al. [11] studies growth in the deployment of ROAs. By comparing 8 years of historical RPKI data to archived routing data from route collectors, they show that the quality of RPKI data (i.e., to what extent ROAs match BGP advertisements) has improved dramatically. 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. [68] also look at this and come up with a solution that uses threshold signatures to limit the power of trust roots. Others have come up with solutions that involve the Blockchain [77] to counter the centralised nature of RPKI. Additionally, RFC 7132 [27] mentions some security considerations regarding an adversarial CA, e.g., "An attacker could create very deep subtrees with many ROAs per publication point, ...", but does not assess the impact in detail, nor does it create a comprehensive analysis of threats to RPKI itself.

Regarding communication with RPKI repositories, there is little public security research into RRDP. The RRDP specification [6] does include security considerations, but those mostly concern integrity and tampering, with references to the recommendations for the use of TLS [67] and some additional notes on how RRDP increases reliability.

4 THREAT MODEL

In this section, we introduce our threat model. The goal of our model is to identify *availability* threats to the RPKI, particularly threats that a dishonest CA can pose to the availability of relying party software. Like we mentioned in Section 2, a lot of known availability problems already exist for rsync on the "server" side — we now swap it around. If an attacker succeeds in disrupting RP software, this may lead to BGP route hijacks succeeding, as a router is not provided with information on valid route origins. Another outcome may be denial of service, which can prevent new routes or updates from being accepted. If a ROA cannot be validated, parties relying on validation will then reject announcements based on stale information.

For this model, we assume that an attacker operates its own RPKI Certificate Authority and repository publication point as part of an otherwise non-malicious RPKI tree and at the same level as other non-malicious organisations. Attackers could use their own resources for this, but they could also gain access to a trusted company's hardware that a publication point and certificate authority runs on via a method such as social engineering, software vulnerabilities, and others. Such an entity has the ability to sign new CAs and objects. We will first sketch the unwritten expectations made about the RPKI in Section 4.1, and then expand on the implications of breaking those unwritten expectations in Section 4.2. We base our tests on common vulnerabilities found in other protocols and projects, such as the ones described in OWASP [60, 61].

4.1 Unwritten Expectations

Let us look at the expectations that exist about the functioning of the RPKI, and whether these expectations are guarded by technological measures, are described as part of the standards, or are not guarded at all. We observe that three types of expectations typically exist: (1) about the size and depth of the RPKI tree hierarchy, (2) on execution time of the validation process, and (3) on the protocol stack RPKI communication protocols depend upon and the operating system relying party software runs on. We discuss these three classes of expectations below.

4.1.1 Tree Size and Depth.

When one considers the RPKI, the most common visualisation is a forest of trees as depicted in Figure 4. Thus, the first expectation is 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. It must be possible to evaluate each tree in isolation, and trees must no contain loops. Furthermore, the RPKI validation process hinges on first collecting all information from all 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 that routes become unavailable. If we take the

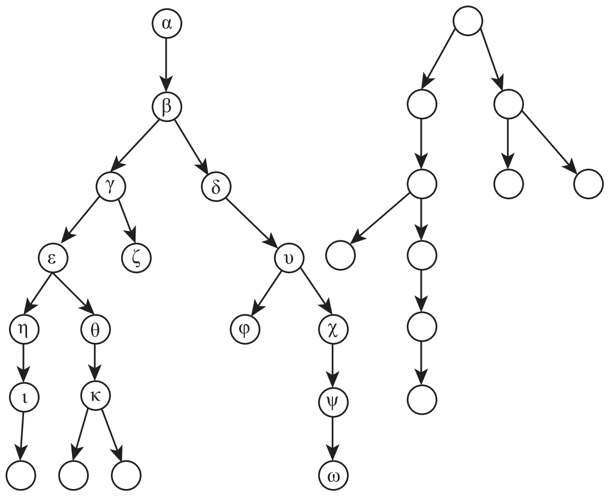


Fig. 4. An example of two RPKI trees. Each arrow signifies a parent-child relationship.

tree in 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).

When we look at the nodes themselves, we make another set of unwritten expectations. Let us look at node υ as shown in Figure 4. We expect 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 (α, β and δ) nor its siblings (γ and further down). Furthermore, a node must 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 all relying parties are served the same data by repositories, and thus have the same view on the RPKI.

Summarising, this means that the RPKI is based on the expectation that the tree is finite and reasonably sized, and that that is the case for every relying party that fetches objects from repositories in the tree.

4.1.2 *Time.*

All of the expectations 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 expected that the tree can be evaluated and processed in a reasonable amount of time, generally considered to be between 30 and 60 minutes [29]. If the process takes significantly longer, objects may expire before they can be evaluated. What is "too long" is undefined, but the shortest lifetime observed for objects in the wild is around 8 hours.

4.1.3 *Protocol Stack and Operating System.*

As stated at the end of Section 2.4, when looking at communication protocols in the RPKI, we focus mostly on RRDP. The fact that RRDP is based on XML and HTTPS means that the threats that apply to that stack — namely, XML, HTTPS, and TCP (or UDP in the case of HTTP/3 [3]) — apply to RRDP as well. RRDP applies some extra restrictions, such as requiring the use of US-ASCII as character set and forbidding the verification of the WebPKI certificate [6] on the TLS or QUIC layer. The signed objects themselves are CMS-encoded X.509 objects [23], which means that 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 typically 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 and other availability threats to the OS apply here as well.

4.2 Implications

Given the expectations discussed in the previous section, we now discuss the implications and build our threat model to analyse threats by a dishonest CA to the availability of relying party software. We break down the threats in the sections below.

4.2.1 HTTPS. As stated before, modern RPKI implementations chiefly use RRDP to fetch objects from repository publication points. RRDP runs over HTTPS and thus potentially imports security issues from this protocol. In particular, we consider the following to be threats in the context of RPKI availability:

- *Decompression bombs* — if the client supports compression such as GZIP, the server may serve a small file that is orders of magnitudes larger when unpacked [9];
- *Redirect loops* — returning a 301 HTTP status code with a location header with a location that either loops or keeps forwarding the client to another location [60];
- *Forcing long retries* — returning a 429 HTTP status code with a retry-after header with an unreasonably high value, such as a year or century [60].

4.2.2 TCP. Until HTTP/3 over QUIC becomes the norm, HTTPS relies on TCP as the underlying transport protocol. This means that RRDP also imports attacks on TCP. In the context of availability, we are most concerned about attacks that attempt to stall connections, such as a Slowloris attack [12] or simulating a very slow unreliable connection.

4.2.3 XML. RRDP uses XML as a message format. This makes it susceptible to attacks that exploit properties of XML [61]. In the context of this work, we consider:

- *Coercive parsing*, where an extreme (possibly endless) depth file is parsed;
- *Invalid structures*, where the XML parser may trip up due to the unconventional nature of the file;
- *Very large payloads*, where the XML parser may load everything into memory or temporarily store everything on disk, causing resource exhaustion;
- *Use of external entities*, thereby requesting resources the client may not be able to access, and possibly forwarding clients to a remote server;
- *Infinite entity expansion*, by specifying recursive entities, or referential entities, an XML parser may run out of space or memory;
- *Using external references* in XML to let the client DDoS another entity.

4.2.4 X.509. RPKI depends on X.509 at multiple levels (transport and content). X.509 is a complex standard, and the RPKI uses only a subset of its features. 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 [38]. Additionally, given the binary nature of this format, unexpected data may cause the application to misbehave [37]. In the context of this work, we did not consider these potential vulnerabilities.

4.2.5 ASN.1. RPKI not only makes use of X.509, but also CMS, both of which are specified using ASN.1 and thus require parsers for objects that follow ASN.1 encodings, such as the Basic and Distinguished Encoding Rules (BER and DER, respectively). In the past, several major implementations, such as OpenSSL [36] and BouncyCastle [39], had vulnerabilities related to parsing these encodings. 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.

Table 1. Version Information for the RP Implementations that Were Tested

	Version	**Operating System**
Routinator	0.10.1	Ubuntu 18.04 LTS
Validator 3	3.2.2021.04.07.12.55	Ubuntu 18.04 LTS
OctoRPKI	1.3.0	Ubuntu 18.04 LTS
Fort	1.5.1	Ubuntu 18.04 LTS
RPKI-Prover	0.1.0	Ubuntu 18.04 LTS
rpstir2	master-7394f73	Ubuntu 18.04 LTS
rpki-client	7.3	OpenBSD 7.0
rcynic	1.0.1544679302	Ubuntu 16.04 LTS

4.2.6 Trees. RPKI repositories are expected 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 these two approaches.

4.2.7 Operating System. Finally, an attacker can attempt to target the underlying file system. Many Linux systems, for example, use `ext4` as filesystem, which has a limited amount of inodes (pointers to entries) available. An attacker could serve a repository with many objects aiming to exhaust the available inodes. Alternatively, an attacker can try to serve extremely large files, e.g., a several gigabyte large Ghostbusters record to exhaust disk space [15].

5 EVALUATING THREATS IN PRACTICE

In the analysis in Section 4, we identified a set of potential vulnerabilities. To test whether current relying party implementations are susceptible to these vulnerabilities, we designed and implemented a testbed that instantiates these vulnerabilities. In this section, we briefly introduce our testbed and discuss the tests we labelled A through O. We note that, in order not to disrupt any relying parties in production, our testbed functions independently from the existing RPKI hierarchy (discussed in more detail in Section 7).

5.1 Testbed

We set our tests up as actual repositories that could appear in the wild. This means we did not need to modify the relying party software implementations. We wanted to ensure that tests could be run independently and reproducibly.

Our testbed creates custom repositories and CAs, and a separate trust anchor (TAL) is available for each test from A through O. The server creates the files for each instance on demand using a randomly generated UUID. This prevents two test instances from interfering with each other and also prevents the results from being 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 hyphen 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. An overview of the setup can be found in Figure 5.

All relying party software implementations were installed on separate virtual machines, with standard configurations according to the instruction manual provided by the implementation. The version numbers of the RP implementations and operating systems of the VMs are listed in Table 1.

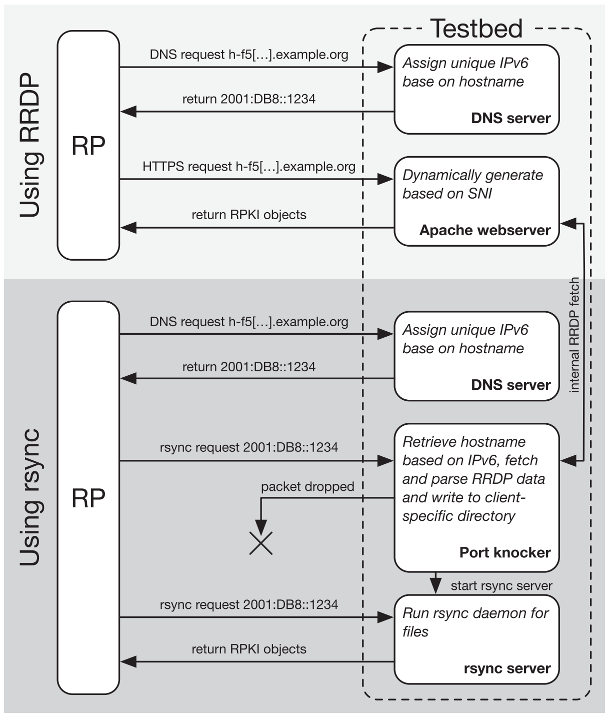

Testbed
Using RRDP
RP
DNS request h-f5[...].example.org
return 2001:DB8::1234
Assign unique IPv6 base on hostname
DNS server
HTTPS request h-f5[...].example.org
return RPKI objects
Dynamically generate based on SNI
Apache webserver
internal RRDP fetch
Using rsync
RP
DNS request h-f5[...].example.org
return 2001:DB8::1234
Assign unique IPv6 base on hostname
DNS server
rsync request 2001:DB8::1234
packet dropped
Retrieve hostname based on IPv6, fetch and parse RRDP data and write to client-specific directory
Port knocker
start rsync server
rsync request 2001:DB8::1234
return RPKI objects
Run rsync daemon for files
rsync server

Fig. 5. A graphical 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. The original rsync request packet is dropped, but relying party (RP) software retries after a timeout.

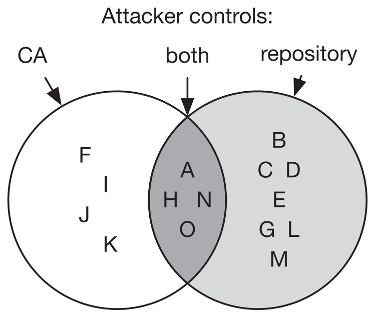


Fig. 6. Attacker control required for specific attacks.

5.2 Tests

We will now discuss the tests we performed based on the identified threats. For clarity, the discussion of each test case is immediately followed by the results of the tests. As discussed earlier in the article, the RPKI distinguishes between the roles of certificate authority and repository Publication Point operator. For the tests we describe, it is often not necessary for an attacker to have both roles. Figure 6 shows a Venn diagram depicting which role(s) an attacker needs to have in order to execute the attacks for each of our test cases.

Table 2. The Tests Executed During Our Research in the Then-Recent Relying Party Software

	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	☒	☒	☒	●	☒	☒	☒	●	☒	☒	☒	☒	☒	☒	☒

● means an implementation was vulnerable. ◐ signifies a vulnerability that was fixed in a version that was released during our study. The crossed-out cells for Validator 3 and rcynic were not tested.

Table 2 summarises the results of the tests per relying party implementation. The details for each test are discussed in the sections that follow. Note that the RIPE NCC RPKI Validator 3 and rcynic were not fully tested. In the case of RIPE NCC RPKI Validator 3, the project was discontinued in July 2021 [65], in the case of rcynic, the last code commit at the time of writing of this article occurred over 6 years ago [17]. We have chosen to include the RIPE NCC RPKI Validator 3 and rcynic because even though they may no longer be supported, they are still used. However, as both have strong indications that there will be no updates for their software, we feel that the message of "your software can be disrupted" is sufficiently strong if we can demonstrate one method to do so rather than several.

Since our disclosure, updates in the various relying party implementations have been released with the aim of resolving the vulnerabilities mentioned.

A. Infinite Repository Chain. We create a chain of repositories, where the repository root has a certificate for a child, and when that child is visited, a certificate for this child is generated. This is repeated *ad infinitum*, creating a never-ending repository chain. This test case tries to attack the expectation that the tree is finite and forces a client that does not impose limits to keep retrieving new data forever.

Result – Most tested implementations keep retrieving data endlessly. Initial tests showed only Fort and rpki-client limit the maximum depth of a repository. A later version of Routinator, which was released while our study was still ongoing, also added a maximum depth.

B. 429 Response Header. The 429 HTTP status code was added in RFC 6585 [18]. 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 again, a "Retry-After" may be included in the response, specifying how many seconds a user should wait before retrying. RFC 8182 [6] does not specify whether this HTTP status code can appear in RRDP, and as relying party software tends to use existing HTTP libraries that may 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 a Retry-After header.

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

C. Endless 302 Response. The 3xx range of HTTP status codes indicates that the resource can be found at another location [56]. These status codes can be chained, i.e., location (1) can link to (2), which links to (3). 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.

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

D. GZIP Bomb. HTTP allows for compression to improve transfer speed [57]. This means that the amount of data transferred may be less than the decompressed size on 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 resources of the system the relying party software runs on. Support for compression is generally explicitly enabled in the HTTP libraries used by relying party implementations.

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

E. Open Connection. Repository publication points 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 expectation by severely restricting transfer speed, for example, to 3 bytes per second. This potentially makes the process of retrieving data run for several weeks if relying parties wait for completion. Note that data is constantly transmitted, just at a very slow rate, to avoid middleboxes cutting off the connection.

Result – All tested implementations apart from RPKI-Prover keep waiting forever. RPKI-Prover imposes a maximum transfer duration, and continues with the next repository once this timer expires.

F. Broken ROA. Whenever relying party software encounters a ROA — a file with a `.roa` extension — it expects this file to have the structure that belongs to a ROA. However, it is possible to encode any data and give it a `.roa` extension. For this test case, the ROA merely consists of an encoded ASCII `NUL` character. A lack of proper input validation may cause relying party software to malfunction or crash when they encounter this broken ROA.

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

G. Billion Laughs Attack. The Billion Laughs attack is an XML expansion attack, where an entity "lol1" is defined as "lol", and "lol2" as ten times "lol1". This is done ten times to create a billion entities [61]. The textual form is small, but XML parsers that try to handle this in memory will likely run out of memory and crash.

Result – Only RPKI-Prover and rpki-client $\leq$ 7.2 crash when encountering the billion laughs attack. All other tested implementations treat this as a broken repository, and retry it over rsync.

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 a reasonable time.

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

I. Impossible ROA. This test is similar to test F, but instead of a broken ROA, we present a syntactically correct ROA that is semantically broken. We do this by undermining the expectations 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 [41], and we expect a similar impact: either the validator crashes, or the output becomes nonsensical, causing the RPKI to Router (RTR) protocol, which facilitates communication between the relying party software and the router, to terminate.

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

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 [8]. The memory of a router is limited. If one were to create ROAs for a prefix for every possible 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 of most current routers.

Result — 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. In all cases, however, we were able to generate a problematic number of entries. The aforementioned safeguards of some RP implementations can likely be circumvented by, for example, delegating part of the ASNs to children or using rsync exclusively.

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 be generated deterministically on the fly; only the SHA-256 hashes need to be stored at 32 bytes per file.

Result — All tested implementations do not limit the maximum number of subprefixes for a prefix; rpki-client exits with "excessive runtime (3600 seconds), giving up". That error message comes up after around 4 to 5 hours rather than the expected 1 hour. We are unsure of what causes this.

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.

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 the URI attribute for the RRDP snapshot as a 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 random bytes as content.

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

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 [35], 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 file system, and pass them to our server, by including an XML external entity in the snapshot URI.

Result — All tested implementations do nothing 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.

N. Long Paths. Much like D, G, and L, this test tries to exhaust the memory of the client running the relying party software, 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 naïvely 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.

Result — 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.

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 21, 2015 [66], 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.

Result – Only OctoRPKI, Fort, and rpstir2 allow writing outside the intended destination folder. The other tested implementations reject the path. We observed that default installation instructions make it easy to accidentally (or even intentionally) run OctoRPKI, Fort and rpstir2 as the root user. This means that a file such as `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 of actively executing the attacks discussed in the previous section as well as their 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 public point 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 [6], 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 difficult. In the case of most RIRs, it consists of a small fee and an identity check [64] to register as a Local Internet Registry (LIR) with its own resource allocation for which the registrant controls RPKI data and can request delegated RPKI.

Parents, such as RIRs and NIRs, can monitor the behaviour of delegated CAs, and revoke a certificate when malicious behaviour is discovered. The main problem with such an approach is that there is no guarantee that a repository serves the same content to all parties. Malicious parties can purposefully exclude the parent from their attack or direct their attack at only one victim based on the IP address or ASN of the relying party client. 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 — it also requires policy development.

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, in which the software would crash when encountering a BGPsec certificate [55], we observed that roughly 3 out of 10 instances did not come back online within a month. Without adequate monitoring, we would expect the long term effects of our attacks to 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 the vulnerabilities we identified have now been resolved after a coordinated vulnerability disclosure to relying party software implementers that ended with public disclosure on November 9, 2021 (discussed in Section 7) [42–50, 53].

7 ETHICAL CONSIDERATIONS

We realised from the start of our study that it was possible that we would discover exploitable vulnerabilities. For this reason, 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 on the "production" RPKI hierarchy, but rather to create our own purpose-built tree hierarchy (as described in Section 5.1).

Over the course of the study, it became clear that we had identified several exploitable vulnerabilities. To address these, we started a multi-party coordinated vulnerability disclosure (CVD) process with the help of the National Cyber Security Centre of the Netherlands (NCSC-NL), where the NCSC-NL coordinated the CVD. We decided to enlist the help of the NCSC-NL as relying party implementers are from all over the world and coordinating such a multi-party process requires specific expertise. The NCSC-NL is an international frontrunner in developing the CVD process [54].

Multi-party disclosures are always difficult processes for which it is necessary to actively coordinate between all of the different stakeholders. For that reason, the NCSC-NL decided to only contact implementers of actively maintained relying party software, using the following selection criteria:

- The implementer is open to confidential multi-party coordinated vulnerability disclosure [54].
- The implementer has published clear points of contact to report security issues to.

At the time of starting the CVD process, the parties included were (1) Routinator by NLnet Labs, (2) RPKI Validator 3 by the RIPE NCC, (3) OctoRPKI by Cloudflare, and (4) Fort Validator by NIC México. The NCSC-NL explicitly decided not to contact the developers of rpki-client, from the OpenBSD project, due to their publicly announced policy of full disclosure of vulnerabilities [58] and due to earlier issues with disclosures such as KRACK [74]. We note that this decision sparked controversy with the OpenBSD developers, which was eventually resolved in discussions between the NCSC-NL and the OpenBSD development team. To enable the OpenBSD team to also resolve the identified vulnerabilities prior to public disclosure, the publication date was postponed by the NCSC-NL. For further information, we refer to the full timeline of the CVD included in Appendix 7.1. At the start of the process (August 4, 2021) the four RP implementations that were contacted by NCSC-NL together represented 97.5% of active RP clients according to the data from the RIPE NCC. Over the course of the CVD process, we used data provided as a courtesy by the RIPE NCC, shown in Figure 7, 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. On the date of disclosure (November 9, 2021), these four implementations together with rpki-client were used by 98.5% of active RP clients according to the data from the RIPE NCC.

Other projects that were not contacted at the start of the CVD process because they did not meet the criteria set by the NCSC-NL 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.

We note that all developers of relying party software (also those not included in the CVD process by NCSC-NL) were notified one week prior to public disclosure, and were given the opportunity to respond and request

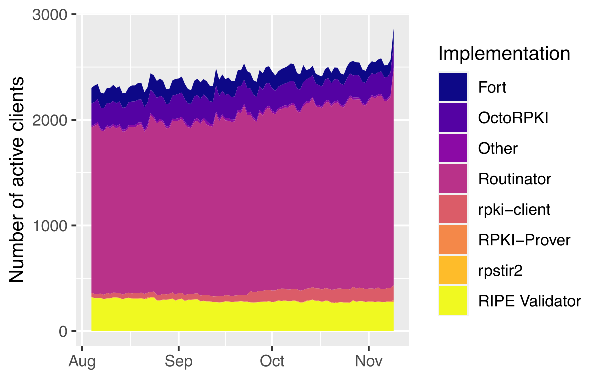


Fig. 7. Absolute relying party usage as observed by the RIPE NCC in the months leading up to the disclosure date. All dates are in 2021. Instances of rpki-client on the NLNOG ring are excluded from the count, as these instances are installed by the developers of rpki-client for automated integration testing.

an extension of the publication deadline. The developer of RPKI-Prover implemented fixes but did not request a deadline extension, the developers of rpstir2 did not respond to any messages, and the developers of rcynic also did not request an extension nor indicate that they were intending to implement patches. At the time of submission of this article, neither rpstir2 nor rcynic have updated their software to protect against the issues we identified and disclosed to them.

7.1 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.
12 August	RIPE NCC responds that they 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	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, with notification of all other parties (rpki-client, rcynic, rpstir2, and RPKI-Prover) set for October 27.
26 October, 16:00 UTC	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, 18:00 UTC	OctoRPKI pull request is removed from the public record.

26 October, 19:00 UTC	Decision is made to move notification of other parties forward by a few hours in light of the public PR.
26 October, 19:00 UTC	Notifications are sent to rpki-client, rcynic, rpstir2, and RPKI-Prover. These notifications are sent by the researcher instead of the 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 agree on an entirely new deadline provided that 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.
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 [52].
30 October	OpenBSD changes its position and agrees to keep the disclosed information confidential under embargo and is informed of the vulnerabilities.
31 October	On request by the OpenBSD developers, the disclosure date is moved to 9 November.
9 November, 14:00h UTC	Embargo is lifted, updates released, CVD process ends.

7.2 Lessons Learned from the Disclosure Process

We noticed that from the outside, it was difficult to distinguish the roles the NCSC-NL and the University of Twente had during the disclosure process. The NCSC-NL orchestrated the disclosure process, whereas the University of Twente handled the technical details of the vulnerabilities found. We found that during our process, we did not always make this distinction clear, thereby seemingly creating confusion regarding the responsibilities from the other parties.

8 SOLVING VULNERABILITIES

During our research, we found several security vulnerabilities that would allow a malicious certificate authority and/or publication point to disrupt RPKI relying party software. Our aim is to make sure that users are no longer vulnerable to these issues, which means that the vulnerabilities need to be resolved on all levels. As we described in Section 7, we made an attempt to resolve these issues on the software side by reporting the vulnerabilities to the software vendors.

8.1 Package Maintainers

Our findings were published in November 2021. We compare the period immediately after the disclosure (up until January 1, 2022) and the situation one year later (up until January 1, 2023).

As of January 1, 2022, insecure versions of relying party software are still present in the default repositories of many operating system distributions. Ongoing data provided by Kristoff et al. [29] 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 and 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 January 1, 2022. Further data provided by Kristoff et al. [30] seems to back this data up, showing a large number of vulnerable relying party clients visiting the beacon's endpoint at the start of 2022.

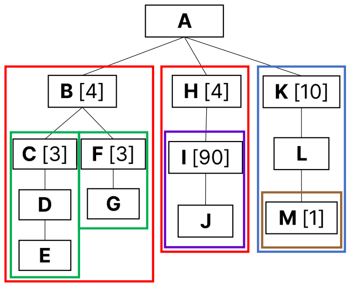
A year later, RIPE NCC RPKI Validator 3 usage has approximately halved based on data by Kristoff et al. [30]. Our own data seems to confirm the current usage figures. Additionally, Debian 11 now features Fort 1.5.3 instead of 1.5.0 and OctoRPKI 1.4.2 instead of 1.2.2. In the latter case, the version is still vulnerable to path traversal attacks (Section O), which were only fully resolved in 1.4.3. We have discussed this with the Debian maintainer of the RPKI packages, but the velocity of RPKI software development makes it difficult for Debian to keep up with.

8.2 IETF

Additionally, we want to draw attention to the fact that all implementations are still vulnerable to test H (exponential repository expansion, see Section 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 does not. Once a parent creates 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 reasons 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. We have submitted a draft to the IETF SIDROPS working group to add the latter [76]. This draft would allow a parent to provide "hints" about what limits should apply to their children, an example of which is shown in Figure 8. This is done by adding a new non-critical extension to the child's CA certificate. In our testing, we found that all current implementations support the addition of new non-critical extensions, meaning that it can be implemented gradually in a backwards compatible way. We shared our ideas with the IETF SIDROPS working group, and whilst there appeared to be some interest initially, a significant part of the discussion was about whether the issue needed solving in the first place. Alternatives were proposed and in part implemented, which according to our models would not prevent abuse of overly large trees. Consensus seems to be that it is unlikely to happen in real life, and that it can be resolved manually when it happens. The tree hints draft has since expired and there seems to be no momentum to revisit it any time soon.

8.3 Router Vendors

We received similar responses when we reported the issues raised in Sections J and K to router vendors such as Cisco and Juniper. Cisco does not expect it to happen in practice. Juniper has confirmed that it would crash their



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

Fig. 8. An example of how RPKI tree hints could work. Shown is an example tree, with the maximum descendants allowed for each node indicated as a number in 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.

routing daemon. They would look into it and follow up with us but after more than 6 months later have not yet done so.

8.4 Organisations

So far, we have seen that both on the software side and the software distribution side, resolving these issues has not always been given priority. We now want to shift our focus to the software that has been updated, both the software itself and the version available in the software repository. After all, even though the latest version of the software may no longer be vulnerable, an organisation may still run an outdated version. To analyse this, we look only at the Routinator versions we have seen hit our publication point between January 15 and January 25, 2023. This is because Routinator is only available from a repository operated by its implementer, NLnet Labs, meaning that it is unlikely that an outdated version is listed as the newest version. We only look at unique /24 IPv4 subnets or /48 IPv6 subnets that visited the publication point every day during this time period in order to filter out non-production instances.

We see the graph of different versions per unique /24 or /48 in Figure 9. The vulnerabilities we reported were resolved in version 0.10.2. We have not included versions that do not announce themselves as “Routinator" in the user agent as we cannot with certainty say that those are Routinator instances, however, we have a strong suspicion that there are approximately 50 instances currently out in the wild that are older than the oldest version we have listed above.

However, absolute numbers do not tell the entire story, as the impact of a large network or organisation will be far greater than the impact of a small organisation. We have looked at the organisations that run at least one vulnerable version of Routinator on their network based on their ASN by using MaxMind’s GeoLite2 ASN database. We have then linked this information to their AS cone size as provided by CAIDA. The resulting division can be seen in Figure 10. Over 25% remain vulnerable to the vulnerabilities we found, and over half remain vulnerable to a vulnerability fixed in 0.11.3 [51].

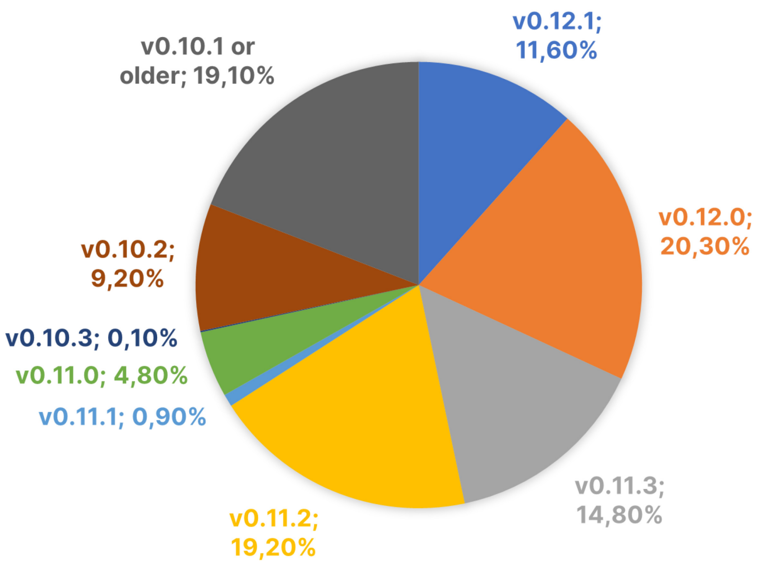


Fig. 9. The version division per unique /24 IPv4 subnets or /48 IPv6 subnets that visited the publication point every day during a 10-day time period in January 2023.

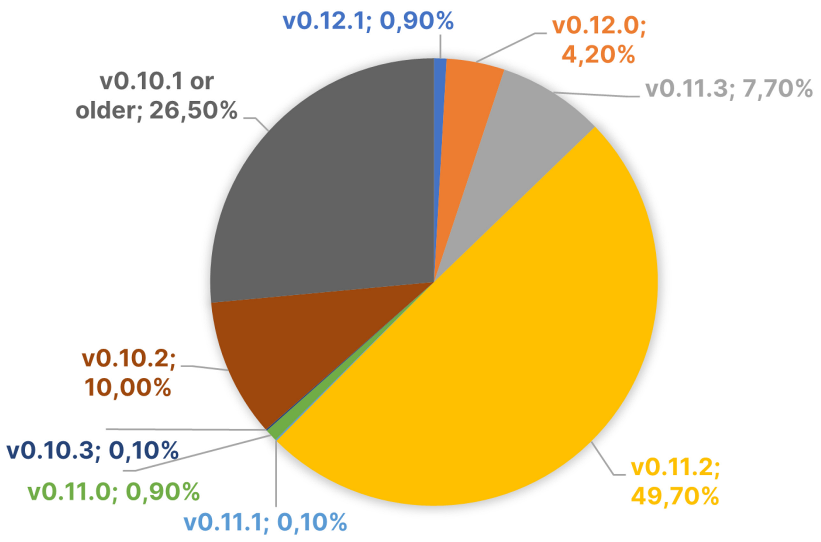


Fig. 10. The version division as in Figure 9 normalised by the size of their AS cone. After a year, still over 25% of the ASs remains vulnerable.

9 CONCLUSION

In this work, we set out to study to what extent a dishonest RPKI certificate authority or repository publication point could disrupt the operation, and specifically the availability, of relying parties. To do this, we created a threat model and analysed the RPKI specifications and underlying communication protocols. Based on this analysis, we identified 15 potential attack vectors, which we implemented in a test bench that can be used to evaluate whether a relying party software implementation is vulnerable to these attacks.

Extensive testing with eight implementations of relying party software showed that all implementations were vulnerable to at least one of the identified attack vectors. We engaged in a coordinated vulnerability disclosure process with the help of the National Cyber Security Centre of the Netherlands (NCSC-NL) to mitigate these vulnerabilities at the vendor side. We have actively engaged with the community to resolve these issues on all levels.

FUTURE WORK

We note that there is a general lack of security analysis for the RPKI, as is the case for many other internet protocols. Given the critical nature of the RPKI for the stability of the internet's routing substrate, we believe it necessary to perform a thorough analysis of other aspects of the RPKI, including proposed future extensions that would allow (partial) path validation [2]. Additionally, we believe it would be of value to look into what the prerequisites would be, and how difficult it would be to meet those, to attack a target organisation using the methods described, especially in a manner such that the perpetrator of the attack is unclear.

ACKNOWLEDGMENTS

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.

CODE AVAILABILITY

The code for the test framework and the manual to set it up are available under the Affero General Public License version 3 [73].

REFERENCES

[1] Donald E. Eastlake 3rd. 2011. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066. (Jan. 2011). https://doi.org/10.17487/RFC6066

[2] Alexander Azimov, Eugene Bogomazov, Randy Bush, Keyur Patel, and Job Snijders. 2021. *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. https://datatracker.ietf.org/doc/html/draft-ietf-sidrops-aspa-verification-07 Work in Progress.

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

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

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

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

[7] Randy Bush. 2012. The Resource Public Key Infrastructure (RPKI) Ghostbusters Record. RFC 6493. (Feb. 2012). https://doi.org/10.17487/RFC6493

[8] Randy Bush and Rob Austein. 2017. The Resource Public Key Infrastructure (RPKI) to Router Protocol, Version 1. RFC 8210. (Sept. 2017). https://doi.org/10.17487/RFC8210

[9] Margaux Canet, Amrit Kumar, Cédric Lauradoux, Mary-Andréa Rakotomanga, and Reihaneh Safavi-Naini. 2017. Decompression quines and anti-viruses. *Proceedings of the 7th ACM Conference on Data and Application Security and Privacy* (Mar 2017). https://doi.org/10.1145/3029806.3029818

[10] Shinyoung Cho, Romain Fontugne, Kenjiro Cho, Alberto Dainotti, and Phillipa Gill. 2019. BGP hijacking classification. In *2019 Network Traffic Measurement and Analysis Conference (TMA).* 25–32. https://doi.org/10.23919/TMA.2019.8784511

[11] Taejoong Chung, Emile Aben, Tim Bruijnzeels, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, Roland van Rijswijk-Deij, John Rula, and Nick Sullivan. 2019. RPKI is coming of age: A longitudinal study of RPKI deployment and invalid route origins. In *Proceedings of the Internet Measurement Conference (IMC'19).* Association for Computing Machinery, 406–419.

[12] Cloudflare. 2021. Slowloris DDoS attack. (2021). https://www.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/

[13] Danny Cooper, Ethan Heilman, Kyle Brogle, Leonid Reyzin, and Sharon Goldberg. 2013. On the risk of misbehaving RPKI authorities. In *Proceedings of the 12th ACM Workshop on Hot Topics in Networks (HotNets-XII)*. Association for Computing Machinery, New York, NY, USA, Article 16, 7 pages. https://doi.org/10.1145/2535771.2535787

[14] Joeri de Ruiter and Erik Poll. 2015. Protocol state fuzzing of TLS implementations. In *24th USENIX Security Symposium (USENIX Security'15)*. USENIX Association, Washington, D.C., 193–206. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter

[15] Borislav Djordjevic and Valentina Timcenko. 2011. 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)*. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA, 288–293.

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

[17] Dragon Research Labs. 2022. rcynic Github Repository. (2022). https://github.com/dragonresearch/rpki.net/tree/master/rp/rcynic (last visited 2022-01-28).

[18] Roy T. Fielding and Mark Nottingham. 2012. Additional HTTP Status Codes. RFC 6585. (April 2012). https://doi.org/10.17487/RFC6585

[19] Lixin Gao. 2001. On inferring autonomous system relationships in the Internet. *IEEE/ACM Transactions on Networking* 9, 6 (2001), 733–745. https://doi.org/10.1109/90.974527

[20] Yossi Gilad, Omar Sagga, and Sharon Goldberg. 2017. MaxLength considered harmful to the RPKI. *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies* (Oct 2017). https://doi.org/10.1145/3143361.3143363

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

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

[23] Russ Housley. 2009. Cryptographic Message Syntax (CMS). RFC 5652. (Sept. 2009). https://doi.org/10.17487/RFC5652

[24] Geoff Huston, Robert Loomans, and George G. Michaelson. 2012. A Profile for Resource Certificate Repository Structure. RFC 6481. (Feb. 2012). https://doi.org/10.17487/RFC6481

[25] Geoff Huston and George G. Michaelson. 2012. Validation of Route Origination Using the Resource Certificate Public Key Infrastructure (PKI) and Route Origin Authorizations (ROAs). RFC 6483. (Feb. 2012). https://doi.org/10.17487/RFC6483

[26] Internet Engineering Task Force. 2023. Introduction to the IETF. (2023). https://www.ietf.org/about/introduction/

[27] Stephen Kent and Andrew Chi. 2014. Threat Model for BGP Path Security. RFC 7132. (Feb. 2014). https://doi.org/10.17487/RFC7132

[28] Stephen Kent, Geoff Huston, and George G. Michaelson. 2012. Certification Authority (CA) Key Rollover in the Resource Public Key Infrastructure (RPKI). RFC 6489. (Feb. 2012). https://doi.org/10.17487/RFC6489

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

[30] John Kristoff, Bill Eaheart, and Matthew P. Kemp. 2022. Aging relying party clients. (May 2022). https://dataplane.substack.com/p/aging-relying-party-clients

[31] Matt Lepinski and Stephen Kent. 2012. An Infrastructure to Support Secure Internet Routing. RFC 6480. (Feb. 2012). https://doi.org/10.17487/RFC6480

[32] Matt Lepinski, Stephen Kent, Geoff Huston, and Rob Austein. 2012. Manifests for the Resource Public Key Infrastructure (RPKI). RFC 6486. (Feb. 2012). https://doi.org/10.17487/RFC6486

[33] Matt Lepinski, Derrick Kong, and Stephen Kent. 2012. A Profile for Route Origin Authorizations (ROAs). RFC 6482. (Feb. 2012). https://doi.org/10.17487/RFC6482

[34] Matt Lepinski and Kotikalapudi Sriram. 2017. BGPsec Protocol Specification. RFC 8205. (Sept. 2017). https://doi.org/10.17487/RFC8205

[35] Mitre. 2015. CVE-2015-3451. (2015). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3451

[36] Mitre. 2016. CVE-2016-2108. (2016). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-2108

[37] Mitre. 2016. CVE-2016-6308. (2016). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6308

[38] Mitre. 2018. CVE-2018-1000613. (2018). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1000613

[39] Mitre. 2019. CVE-2019-17359. (2019). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-17359

[40] Mitre. 2019. CVE-2019-6659. (2019). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-6659

[41] Mitre. 2021. CVE-2021-3761. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3761

[42] Mitre. 2021. CVE-2021-3907. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3907

[43] Mitre. 2021. CVE-2021-3908. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3908

[44] Mitre. 2021. CVE-2021-3909. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3909

[45] Mitre. 2021. CVE-2021-3910. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3910
[46] Mitre. 2021. CVE-2021-3911. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3911
[47] Mitre. 2021. CVE-2021-3912. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-3912
[48] Mitre. 2021. CVE-2021-43172. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43172
[49] Mitre. 2021. CVE-2021-43173. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43173
[50] Mitre. 2021. CVE-2021-43174. (2021). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-43174
[51] Mitre. 2022. CVE-2022-3029. (2022). http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-3029
[52] Nationaal Cyber Security Centrum. 2021. Upcoming Announcement of RPKI CVD Procedure. (Oct 2021). https://web.archive.org/web/20211029165109/https://english.ncsc.nl/latest/news/2021/october/29/upcoming-announcement-of-rpki-cvd-procedure
[53] Nationaal Cyber Security Centrum. 2021. NCSC-2021-0987. (2021). https://www.ncsc.nl/actueel/advisory?id=NCSC-2021-0987
[54] National Cyber Security Centre of the Netherlands. 2018. *Coordinated Vulnerability Disclosure: The Guideline*. Technical Report. Ministry of Justice and Security. https://english.ncsc.nl/get-to-work/publications/publications/2019/juni/01/coordinated-vulnerability-disclosure-the-guideline
[55] NICMx/FORT-validator. 2021. 1.5.3 crashing repeatedly starting about an hour ago. (2021). https://github.com/NICMx/FORT-validator/issues/65
[56] Henrik Nielsen, Roy T. Fielding, and Tim Berners-Lee. 1996. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945. (May 1996). https://doi.org/10.17487/RFC1945
[57] Henrik Nielsen, Jeffrey Mogul, Larry M. Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. 1999. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. (June 1999). https://doi.org/10.17487/RFC2616
[58] OpenBSD. 2021. OpenBSD Security. (2021). https://www.openbsd.org/security.html
[59] OpenBSD. 2021. OpenRsync. (2021). https://www.openrsync.org/
[60] OWASP. 2021. REST Security Cheat Sheet. (2021). https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html
[61] OWASP. 2021. XML Security Cheat Sheet. (2021). https://cheatsheetseries.owasp.org/cheatsheets/XML_Security_Cheat_Sheet.html
[62] Yakov Rekhter, Susan Hares, and Tony Li. 2006. A Border Gateway Protocol 4 (BGP-4). RFC 4271. (Jan. 2006). https://doi.org/10.17487/RFC4271
[63] Eric Rescorla. 2018. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. (Aug. 2018). https://doi.org/10.17487/RFC8446
[64] RIPE NCC. 2021. Become a RIPE NCC Member. (2021). https://www.ripe.net/participate/member-support/become-a-member
[65] RIPE NCC. 2021. Ending Support for the RIPE NCC RPKI Validator. (2021). https://www.ripe.net/publications/news/announcements/ending-support-for-the-ripe-ncc-rpki-validator
[66] Rsync. 2021. Rsync Security Advisories. (2021). https://rsync.samba.org/security.html
[67] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. 2015. Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7525. (May 2015). https://doi.org/10.17487/RFC7525
[68] Kris Shrishak and Haya Shulman. 2020. Limiting the power of RPKI authorities. In *Proceedings of the Applied Networking Research Workshop (ANRW'20)*. Association for Computing Machinery, New York, NY, USA, 12–18. https://doi.org/10.1145/3404868.3406674
[69] Tom Strickx. 2019. How Verizon and a BGP Optimizer Knocked Large Parts of the Internet Offline Today. (Jun 2019). https://blog.cloudflare.com/how-verizon-and-a-bgp-optimizer-knocked-large-parts-of-the-internet-offline-today/
[70] Andree Toonk. 2017. Popular destinations rerouted to Russia. (Dec 2017). https://bgpmon.net/popular-destinations-rerouted-to-russia/
[71] Nathalie Trenaman. 2021. Update on RIPE Roadmap and Experiences. (2021). https://ripe82.ripe.net/archives/video/602/ RIPE 82.
[72] Iljitsch van Beijnum. 2008. Insecure routing redirects YouTube to Pakistan. (Feb 2008). https://arstechnica.com/uncategorized/2008/02/insecure-routing-redirects-youtube-to-pakistan/
[73] Koen van Hove. 2022. Relying party resiliency platform. (Aug 2022). https://doi.org/10.5281/zenodo.7572512
[74] Mathy Vanhoef. 2017. Key Reinstallation Attacks - Breaking WPA2 by forcing nonce reuse. (2017). https://www.krackattacks.com/
[75] Matthias Wählisch, Robert Schmidt, Thomas C. Schmidt, Olaf Maennel, Steve Uhlig, and Gareth Tyson. 2015. 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)*. Association for Computing Machinery, New York, NY, USA, Article 11, 7 pages. https://doi.org/10.1145/2834050.2834102
[76] Koen van Hove. 2021. *Tree Hints for the Resource Public Key Infrastructure (RPKI)*. Internet-Draft. Internet Engineering Task Force. withheldfordouble-blindreview Work in Progress.
[77] Zhiwei Yan and Jong-Hyouk Lee. 2021. BGPChain: Constructing a secure, smart, and agile routing infrastructure based on blockchain. *ICT Express* (2021). https://doi.org/10.1016/j.icte.2020.12.005

Received 27 March 2023; revised 15 June 2023; accepted 3 August 2023