Stream: Internet Engineering Task Force (IETF)
RFC: 9618
Updates: 5280
Category: Standards Track
Published: August 2024
ISSN: 2070-1721
Author: D. Benjamin
*Google LLC*

# RFC 9618
# Updates to X.509 Policy Validation

## Abstract

This document updates RFC 5280 to replace the algorithm for X.509 policy validation with an equivalent, more efficient algorithm. The original algorithm built a structure that scaled exponentially in the worst case, leaving implementations vulnerable to denial-of-service attacks.

## Status of This Memo

## Copyright Notice

## Table of Contents

## 1. Introduction

[RFC5280] defines a suite of extensions for determining the policies that apply to a certification path. A policy is described by an object identifier (OID) and a set of optional qualifiers.

Policy validation in [RFC5280] is complex. As an overview, the certificate policies extension (Section 4.2.1.4 of [RFC5280]) describes the policies, with optional qualifiers, under which an individual certificate was issued. The policy mappings extension (Section 4.2.1.5 of [RFC5280]) allows a CA certificate to map its policy OIDs to other policy OIDs in certificates that it issues. Subject to these mappings and other extensions, the certification path's overall policy set is the intersection of policies asserted by each certificate in the path.

The procedure in Section 6.1 of [RFC5280] determines this set in the course of certification path validation. It does so by building a policy tree containing policies asserted by each certificate and the mappings between them. This tree can grow exponentially in the depth of the certification path, which means an attacker, with a small input, can cause a path validator to consume excessive memory and computational resources. This cost asymmetry can lead to a denial-of-service vulnerability in X.509-based applications, such as [CVE-2023-0464] and [CVE-2023-23524].

Section 3 describes this vulnerability. Section 4.1 describes the primary mitigation for this vulnerability, a replacement for the policy tree structure. Section 5 provides updates to [RFC5280] that implement this change. Finally, Section 6 discusses alternative mitigation strategies for X.509 applications.

### 1.1. Summary of Changes from RFC 5280

The algorithm for processing certificate policies and policy mappings is replaced with one that builds an equivalent but much more efficient structure. This new algorithm does not change the validity status of any certification path or which certificate policies are valid for it.

## 2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

## 3. Denial-of-Service Vulnerability

This section discusses how the path validation algorithm defined in Section 6.1.2 of [RFC5280] can lead to a denial-of-service vulnerability in X.509-based applications.

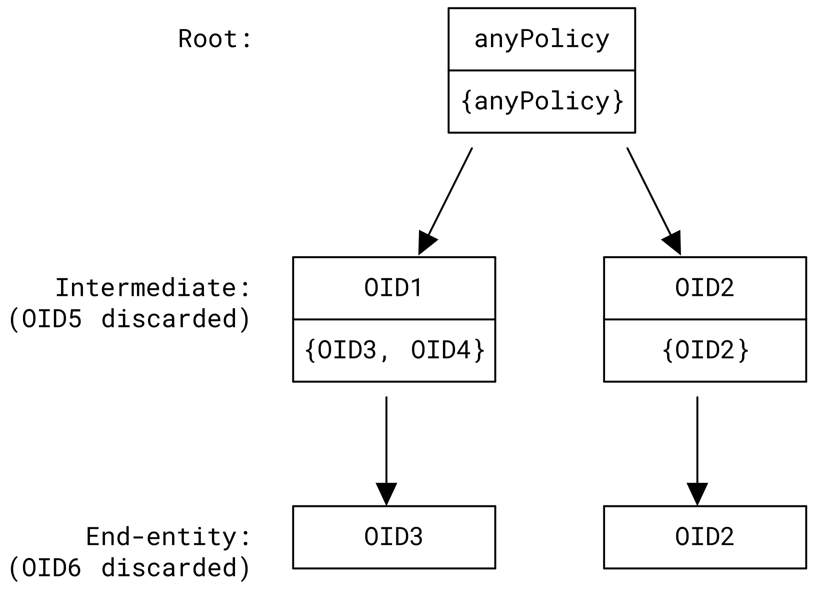
### 3.1. Policy Trees

Section 6.1.2 of [RFC5280] constructs the `valid_policy_tree`, a tree of certificate policies, during certification path validation. The nodes at any given depth in the tree correspond to policies asserted by a certificate in the certification path. A node's parent policy is the policy in the issuer certificate that was mapped to this policy, and a node's children are the policies the node was mapped to in the subject certificate.

For example, suppose a certification path contains:

- An intermediate certificate that asserts the following policy OIDs: OID1, OID2, and OID5. It contains mappings from OID1 to OID3 and from OID1 to OID4.
- An end-entity certificate that asserts the following policy OIDs: OID2, OID3, and OID6.

This would result in the tree shown below. Note that OID5 and OID6 are not included or mapped across the whole path, so they do not appear in the final structure.



The complete algorithm for building this structure is described in steps (d), (e), and (f) in Section 6.1.3 of [RFC5280]; steps (h), (i), and (j) in Section 6.1.4 of [RFC5280]; and steps (a), (b), and (g) in Section 6.1.5 of [RFC5280].

## 3.2. Exponential Growth

The `valid_policy_tree` grows exponentially in the worst case. In step (d.1) in Section 6.1.3 of [RFC5280], a single policy P can produce multiple child nodes if multiple issuer policies map to P. This can cause the tree size to increase in size multiplicatively at each level.

In particular, consider a certificate chain where every intermediate certificate asserts policies OID1 and OID2 and then contains the full Cartesian product of mappings:

- OID1 maps to OID1
- OID1 maps to OID2
- OID2 maps to OID1
- OID2 maps to OID2

At each depth, the tree would double in size. For example, if there are two intermediate certificates and one end-entity certificate, the resulting tree would be as depicted in Figure 1.

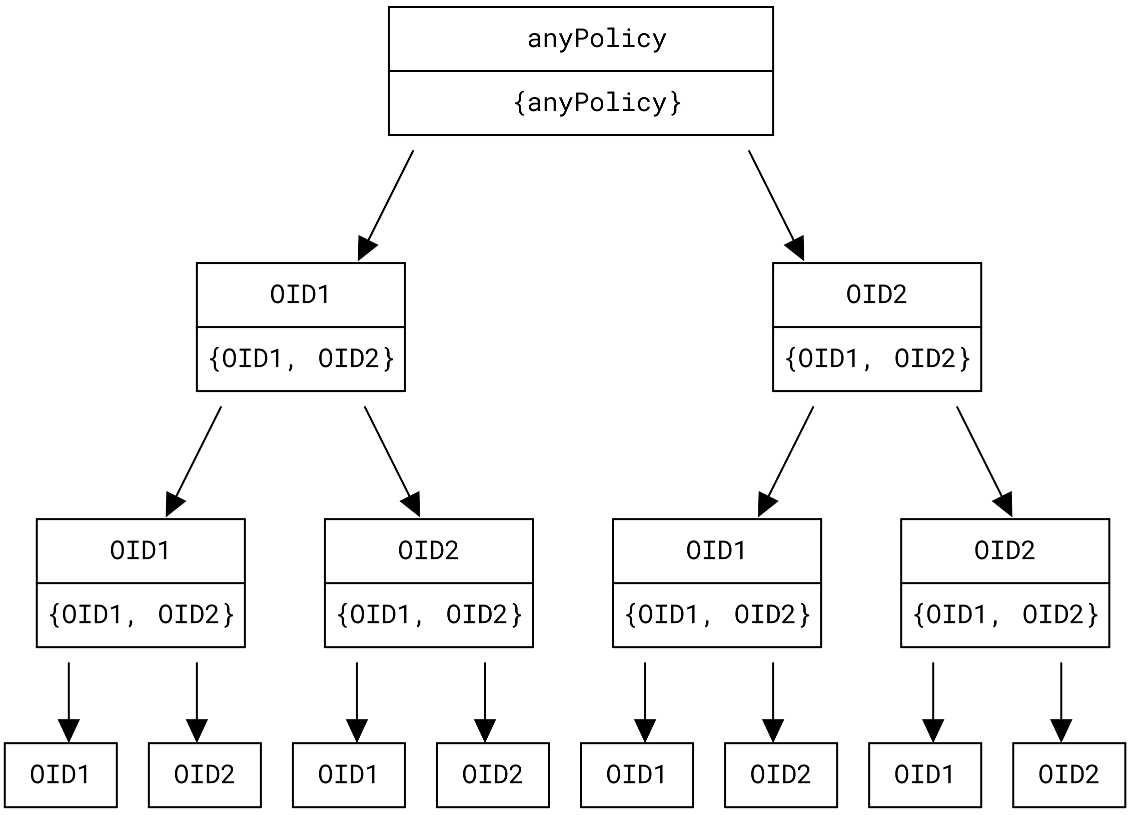


*Figure 1: An Example X.509 Policy Tree with Exponential Growth*

### 3.3. Attack Vector

An attacker can use the exponential growth to mount a denial-of-service attack against an X.509-based application. The attacker sends a certificate chain as described in Section 3.2 and triggers the target application's certificate validation process. For example, the target application may be a TLS server [RFC8446] that performs client certificate validation. The target application will consume far more resources processing the input than the attacker consumed to send it, which prevents the target application from servicing other clients.

## 4. Avoiding Exponential Growth

This document mitigates the denial-of-service vulnerability described in Section 3 by replacing the policy tree with a policy graph structure, which is described in this section. The policy graph grows linearly instead of exponentially. This removes the asymmetric cost in policy validation.

X.509 implementations **SHOULD** perform policy validation by building a policy graph, following the procedure described in Section 5. This replacement procedure computes the same policies as in [RFC5280], but one of the outputs is in a different form. See Section 4.2 for details. Section 6 describes alternative mitigations for implementations that depend on the original, exponential-sized output.

### 4.1. Policy Graphs

The tree structure in [RFC5280] is an unnecessarily inefficient representation of a certification path's policy mappings. When multiple issuer policies map to a single subject policy, the subject policy will correspond to multiple duplicate nodes in the policy tree. Children of the subject policy are then duplicated recursively. This duplication is the source of the exponential growth described in Section 3.2.

A policy graph represents the same information with a directed acyclic graph of policy nodes. It eliminates this duplication by using a single node with multiple parents. See Section 5 for the procedure for building this structure. Figure 2 shows the updated representation of the example in Figure 1.

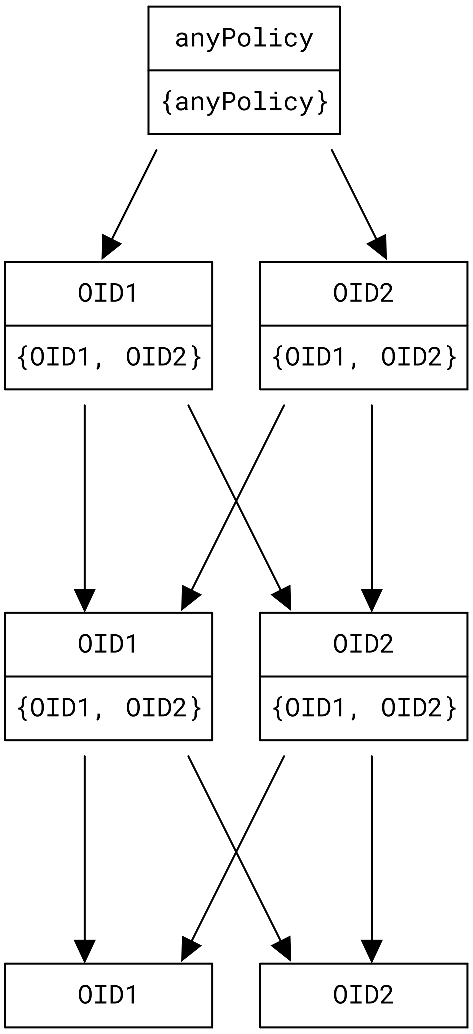


*Figure 2: A More Efficient Representation of an X.509 Policy Tree*

This graph's size is bounded linearly by the total number of certificate policies (Section 4.2.1.4 of [RFC5280]) and policy mappings (Section 4.2.1.5 of [RFC5280]). The policy tree in [RFC5280] is the tree of all paths from the root to a leaf in the policy graph, so no information is lost in the graph representation.

## 4.2. Verification Outputs

Section 6.1.6 of [RFC5280] describes the entire `valid_policy_tree` structure as an output of the verification process. However, Section 12.2 of [X.509] only describes the following as outputs: the authorities-constrained policies, the user-constrained policies, and their associated qualifiers.

As the `valid_policy_tree` is the exponential structure, computing it reintroduces the denial-of-service vulnerability. X.509 implementations **SHOULD NOT** output the entire `valid_policy_tree` structure; instead, they **SHOULD** limit output to just the set of authorities-constrained and/or user-constrained policies, as described in [X.509]. Sections 5.6 and 6 discuss other mitigations for applications where this option is not available.

X.509 implementations **MAY** omit policy qualifiers from the output to simplify processing. Note that Section 4.2.1.4 of [RFC5280] already recommends that certification authorities omit policy qualifiers from policy information terms.

## 5. Updates to RFC 5280

This section provides updates to [RFC5280]. These updates implement the changes described in Section 4.

### 5.1. Updates to Section 6.1

Section 6.1 of [RFC5280] is updated as follows:

OLD:

> A particular certification path may not, however, be appropriate for all applications. Therefore, an application **MAY** augment this algorithm to further limit the set of valid paths. The path validation process also determines the set of certificate policies that are valid for this path, based on the certificate policies extension, policy mappings extension, policy constraints extension, and inhibit anyPolicy extension. To achieve this, the path validation algorithm constructs a valid policy tree. If the set of certificate policies that are valid for this path is not empty, then the result will be a valid policy tree of depth n, otherwise the result will be a null valid policy tree.

NEW:

> A particular certification path may not, however, be appropriate for all applications. Therefore, an application **MAY** augment this algorithm to further limit the set of valid paths. The path validation process also determines the set of certificate policies that are valid for this path, based on the certificate policies extension, policy mappings extension, policy constraints extension, and inhibit anyPolicy extension. To achieve this, the path validation algorithm constructs a valid policy set, which may be empty if no certificate policies are valid for this path.

### 5.2. Updates to Section 6.1.2

The following replaces entry (a) in Section 6.1.2 of [RFC5280]:

(a) `valid_policy_graph`: A directed acyclic graph of certificate policies with their optional qualifiers; each of the leaves of the graph represents a valid policy at this stage in the certification path validation. If valid policies exist at this stage in the certification path validation, the depth of the graph is equal to the number of certificates in the chain that have been processed. If valid policies do not exist at this stage in the certification path validation, the graph is set to NULL. Once the graph is set to NULL, policy processing ceases. Implementations **MAY** omit qualifiers if not returned in the output.

Each node in the `valid_policy_graph` includes three data objects: the valid policy, a set of associated policy qualifiers, and a set of one or more expected policy values.

Nodes in the graph can be divided into depths, numbered starting from zero. A node at depth x can have zero or more children at depth x+1 and, with the exception of depth zero, one or more parents at depth x-1. No other edges between nodes may exist.

If the node is at depth x, the components of the node have the following semantics:

(1) The `valid_policy` is a single policy OID representing a valid policy for the path of length x.

(2) The `qualifier_set` is a set of policy qualifiers associated with the valid policy in certificate x. It is only necessary to maintain this field if policy qualifiers are returned to the application. See Section 6.1.5, step (g).

(3) The `expected_policy_set` contains one or more policy OIDs that would satisfy this policy in the certificate x+1.

The initial value of the `valid_policy_graph` is a single node with `valid_policy` anyPolicy, an empty `qualifier_set`, and an `expected_policy_set` with the single value anyPolicy. This node is considered to be at depth zero.

The graph additionally satisfies the following invariants:

- For any depth x and policy OID P-OID, there is at most one node at depth x whose `valid_policy` is P-OID.
- The `expected_policy_set` of a node whose `valid_policy` is anyPolicy is always {anyPolicy}.
- A node at depth x whose `valid_policy` is anyPolicy, except for the one at depth zero, always has exactly one parent: a node at depth x-1 whose `valid_policy` is also anyPolicy.

- Each node at depth greater than 0 has either one or more parent nodes whose `valid_policy` is not anyPolicy or a single parent node whose `valid_policy` is anyPolicy. That is, a node cannot simultaneously be a child of both anyPolicy and some non-anyPolicy OID.

Figure 3 is a graphic representation of the initial state of the `valid_policy_graph`. Additional figures will use this format to describe changes in the `valid_policy_graph` during path processing.

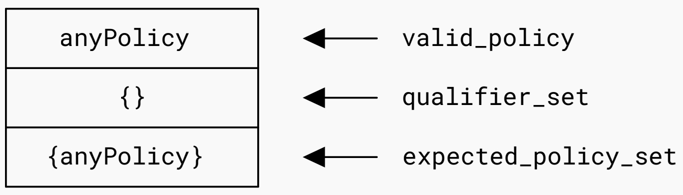


*Figure 3: Initial Value of the `valid_policy_graph` State Variable*

## 5.3. Updates to Section 6.1.3

The following replaces steps (d), (e), and (f) in Section 6.1.3 of [RFC5280]:

(d) If the certificate policies extension is present in the certificate and the `valid_policy_graph` is not NULL, process the policy information by performing the following steps in order:

(1) For each policy P not equal to anyPolicy in the certificate policies extension, let P-OID denote the OID for policy P and P-Q denote the qualifier set for policy P. Perform the following steps in order:

(i) Let `parent_nodes` be the nodes at depth i-1 in the `valid_policy_graph` where P-OID is in the `expected_policy_set`. If `parent_nodes` is not empty, create a child node as follows: set the `valid_policy` to P-OID, set the `qualifier_set` to P-Q, set the `expected_policy_set` to {P-OID}, and set the parent nodes to `parent_nodes`.

For example, consider a `valid_policy_graph` with a node of depth i-1 where the `expected_policy_set` is {Gold, White} and a second node where the `expected_policy_set` is {Gold, Yellow}. Assume the certificate policies Gold and Silver appear in the certificate policies extension of certificate i. The Gold policy is matched, but the Silver policy is not. This rule will generate a child node of depth i for the Gold policy. The result is shown as Figure 4.

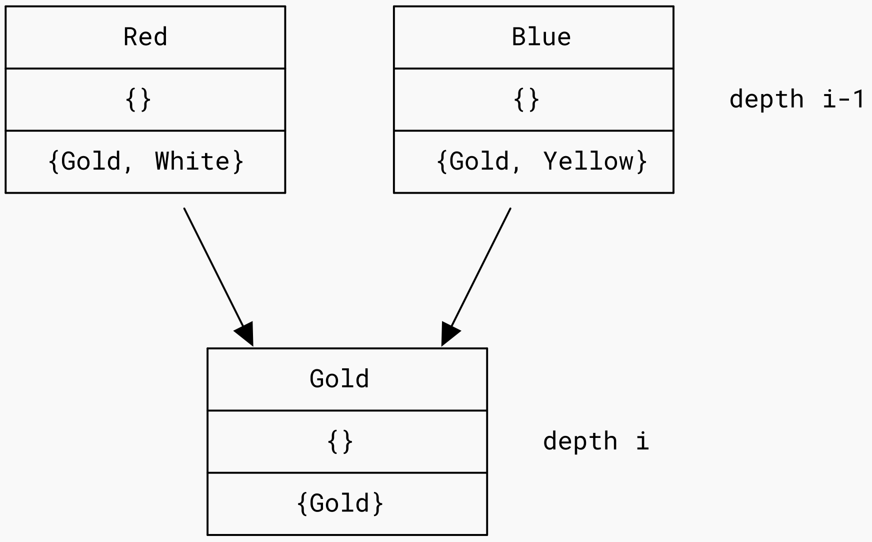


*Figure 4: Processing an Exact Match*

(ii) If there was no match in step (i) and the `valid_policy_graph` includes a node of depth i-1 with the `valid_policy` anyPolicy, generate a child node with the following values: set the `valid_policy` to P-OID, set the `qualifier_set` to P-Q, set the `expected_policy_set` to {P-OID}, and set the parent node to the anyPolicy node at depth i-1.

For example, consider a `valid_policy_graph` with a node of depth i-1 where the `valid_policy` is anyPolicy. Assume the certificate policies Gold and Silver appear in the certificate policies extension of certificate i. The Gold policy does not have a qualifier, but the Silver policy has the qualifier Q-Silver. If Gold and Silver were not matched in (i) above, this rule will generate two child nodes of depth i, one for each policy. The result is shown as Figure 5.

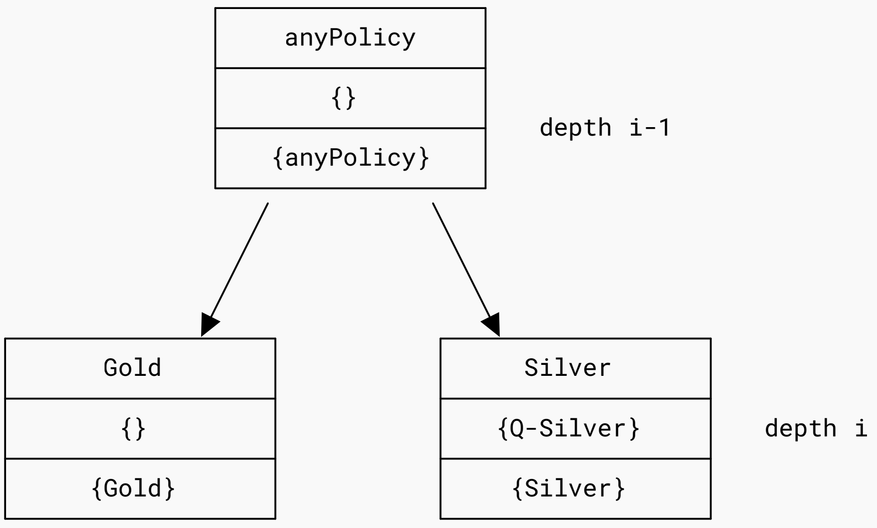


*Figure 5: Processing Unmatched Policies When a Leaf Node Specifies anyPolicy*

(2) If the certificate policies extension includes the policy anyPolicy with the qualifier set AP-Q and either (a) `inhibit_anyPolicy` is greater than 0 or (b) i<n and the certificate is self-issued, then:

For each policy OID P-OID (including anyPolicy) that appears in the `expected_policy_set` of some node in the `valid_policy_graph` for depth i-1, if P-OID does not appear as the `valid_policy` of some node at depth i, create a single child node with the following values: set the `valid_policy` to P-OID, set the `qualifier_set` to AP-Q, set the `expected_policy_set` to {P-OID}, and set the parents to the nodes at depth i-1 where P-OID appears in `expected_policy_set`.

This is equivalent to running step (1) above as if the certificate policies extension contained a policy with OID P-OID and qualifier set AP-Q.

For example, consider a `valid_policy_graph` with a node of depth i-1 where the `expected_policy_set` is {Gold, Silver} and a second node of depth i-1 where the `expected_policy_set` is {Gold}. Assume anyPolicy appears in the certificate policies extension of certificate i with policy qualifiers AP-Q, but Gold and Silver do not appear. This rule will generate two child nodes of depth i, one for each policy. The result is shown below as Figure 6.

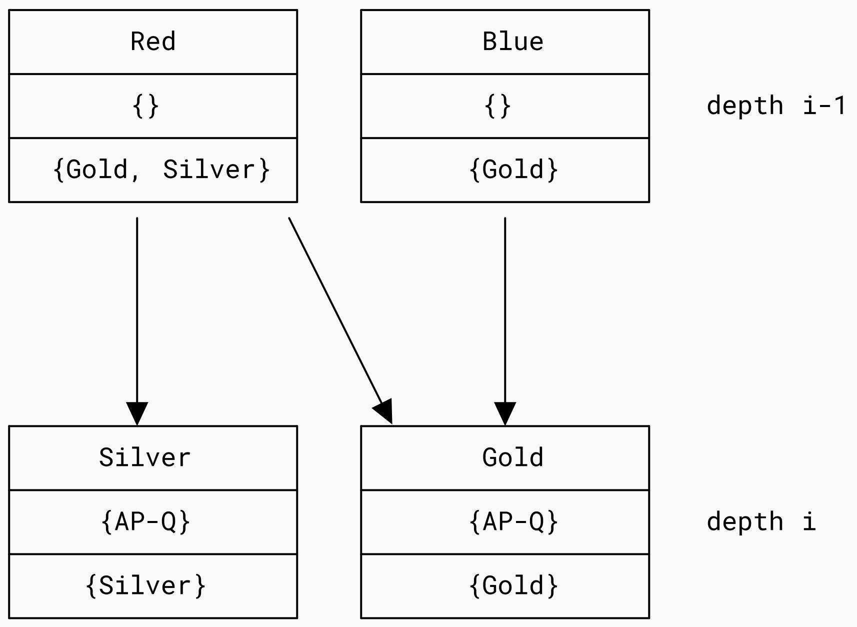


*Figure 6: Processing Unmatched Policies When the Certificate Policies Extension Specifies anyPolicy*

(3) If there is a node in the `valid_policy_graph` of depth i-1 or less without any child nodes, delete that node. Repeat this step until there are no nodes of depth i-1 or less without children.

For example, consider the `valid_policy_graph` shown in Figure 7 below. The two nodes at depth i-1 that are marked with an 'X' have no children, and they are deleted. Applying this rule to the resulting graph will cause the nodes at depth i-2 that is marked with a 'Y' to be deleted. In the resulting graph, there are no nodes of depth i-1 or less without children, and this step is complete.

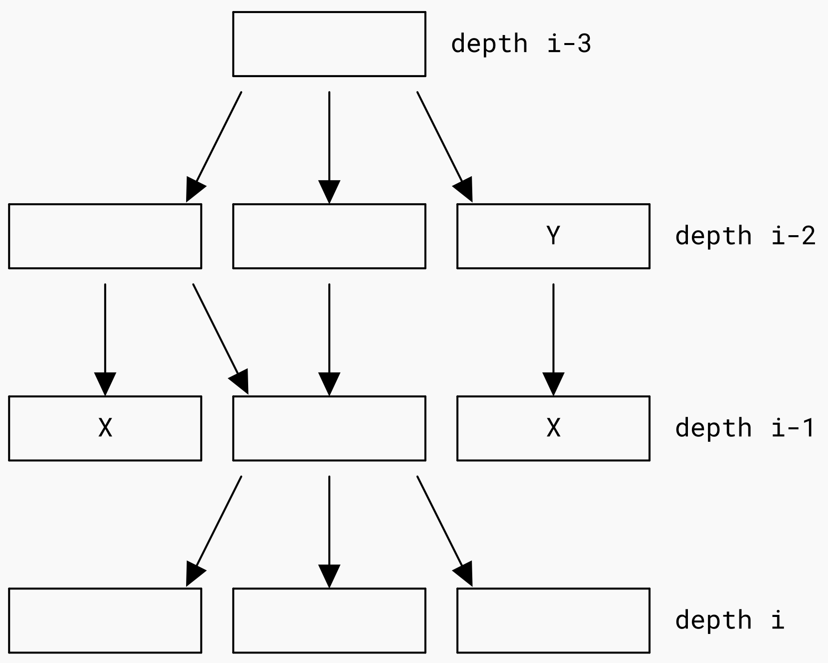


*Figure 7: Pruning the valid_policy_graph*

(e) If the certificate policies extension is not present, set the `valid_policy_graph` to NULL.

(f) Verify that either `explicit_policy` is greater than 0 or the `valid_policy_graph` is not equal to NULL.

The text following step (f) in Section 6.1.3 of [RFC5280], beginning with "If any of steps (a), (b), (c), or (f) fails", is left unmodified.

## 5.4. Updates to Section 6.1.4

The following replaces step (b) in Section 6.1.4 of [RFC5280]:

(b) If a policy mappings extension is present, then for each issuerDomainPolicy ID-P in the policy mappings extension:

(1)

If the policy_mapping variable is greater than 0 and there is a node in the valid_policy_graph of depth i where ID-P is the valid_policy, set expected_policy_set to the set of subjectDomainPolicy values that are specified as equivalent to ID-P by the policy mappings extension.

(2) If the policy_mapping variable is greater than 0 and no node of depth i in the valid_policy_graph has a valid_policy of ID-P but there is a node of depth i with a valid_policy of anyPolicy, then generate a child node of the node of depth i-1 that has a valid_policy of anyPolicy as follows:

  (i) set the valid_policy to ID-P;

  (ii) set the qualifier_set to the qualifier set of the policy anyPolicy in the certificate policies extension of certificate i; and

  (iii) set the expected_policy_set to the set of subjectDomainPolicy values that are specified as equivalent to ID-P by the policy mappings extension.

(3) If the policy_mapping variable is equal to 0:

  (i) delete the node, if any, of depth i in the valid_policy_graph where ID-P is the valid_policy.

  (ii) If there is a node in the valid_policy_graph of depth i-1 or less without any child nodes, delete that node. Repeat this step until there are no nodes of depth i-1 or less without children.

## 5.5. Updates to Section 6.1.5

The following replaces step (g) in Section 6.1.5 of [RFC5280]:

(g) Calculate the user_constrained_policy_set as follows. The user_constrained_policy_set is a set of policy OIDs, along with associated policy qualifiers.

  (1) If the valid_policy_graph is NULL, set valid_policy_node_set to the empty set.

  (2) If the valid_policy_graph is not NULL, set valid_policy_node_set to the set of policy nodes whose valid_policy is not anyPolicy and whose parent list is a single node with valid_policy of anyPolicy.

  (3) If the valid_policy_graph is not NULL and contains a node of depth n with the valid_policy anyPolicy, add it to valid_policy_node_set.

> (4) Compute `authority_constrained_policy_set`, a set of policy OIDs and associated qualifiers as follows. For each node in `valid_policy_node_set`:
>
> (i) Add the node's `valid_policy` to `authority_constrained_policy_set`.
>
> (ii) Collect all qualifiers in the node, its ancestors, and descendants and associate them with `valid_policy`. Applications that do not use policy qualifiers **MAY** skip this step to simplify processing.
>
> (5) Set `user_constrained_policy_set` to `authority_constrained_policy_set`.
>
> (6) If the user-initial-policy-set is not anyPolicy:
>
> (i) Remove any elements of `user_constrained_policy_set` that do not appear in user-initial-policy-set.
>
> (ii) If anyPolicy appears in `authority_constrained_policy_set` with qualifiers AP-Q, for each OID P-OID in user-initial-policy-set that does not appear in `user_constrained_policy_set`, add P-OID with qualifiers AP-Q to `user_constrained_policy_set`.

In addition, the final paragraph in Section 6.1.5 of [RFC5280] is updated as follows:

OLD:

> If either (1) the value of `explicit_policy` variable is greater than zero or (2) the `valid_policy_tree` is not NULL, then path processing has succeeded.

NEW:

> If either (1) the value of `explicit_policy` is greater than zero, or (2) the `user_constrained_policy_set` is not empty, then path processing has succeeded.

## 5.6. Updates to Section 6.1.6

The following replaces Section 6.1.6 of [RFC5280]:

> If path processing succeeds, the procedure terminates, returning a success indication together with the final value of the `user_constrained_policy_set`, the `working_public_key`, the `working_public_key_algorithm`, and the `working_public_key_parameters`.

Note that the original procedure described in [RFC5280] included a `valid_policy_tree` structure as part of the output. This structure grows exponentially in the size of the input, so computing it risks denial-of-service vulnerabilities in X.509-based applications, such as [CVE-2023-0464] and [CVE-2023-23524]. Accordingly, this output is deprecated. Computing this structure is **NOT RECOMMENDED**.

An implementation that requires `valid_policy_tree` for compatibility with legacy systems may compute it from `valid_policy_graph` by recursively duplicating every multi-parent node. This may be done on-demand when the calling application first requests this output. However, this computation may consume exponential time and memory, so such implementations **SHOULD** mitigate denial-of-service attacks in other ways, such as by limiting the depth or size of the tree.

## 6. Other Mitigations

X.509 implementations that are unable to switch to the policy graph structure **SHOULD** mitigate the denial-of-service attack in other ways. This section describes alternate mitigation and partial mitigation strategies.

### 6.1. Verify Signatures First

X.509 validators **SHOULD** verify signatures in certification paths before or in conjunction with policy verification. This limits the attack to entities in control of CA certificates. For some applications, this may be sufficient to mitigate the attack. However, other applications may still be impacted, for example:

- Any application that evaluates an untrusted PKI, such as a hosting provider that evaluates a customer-supplied PKI
- Any application that evaluates an otherwise trusted PKI that includes untrusted entities with technically constrained intermediate certificates. If the intermediates do not constrain policy mapping or path length, those entities may be able to perform this attack.

### 6.2. Limit Certificate Depth

The policy tree grows exponentially in the depth of a certification path, so limiting the depth and certificate size can mitigate the attack.

However, this option may not be viable for all applications. Too low of a limit may reject existing paths that the application wishes to accept. Too high of a limit may still admit a denial-of-service attack for the application. By modifying the example in Section 3.2 to increase the number of policies asserted in each certificate, an attacker could still achieve $O(N^{(depth/2)})$ scaling.

### 6.3. Limit Policy Tree Size

The attack can be mitigated by limiting the number of nodes in the policy tree and rejecting the certification path if this limit is reached. This limit should be set high enough to still admit existing valid certification paths for the application but low enough to no longer admit a denial-of-service attack.

### 6.4. Inhibit Policy Mapping

If policy mapping is disabled via the initial-policy-mapping-inhibit setting (see Section 6.1.1 of [RFC5280]), the attack is mitigated. This also significantly simplifies the X.509 implementation, which reduces the risk of other security bugs. However, this will break compatibility with any existing certification paths that rely on policy mapping.

To facilitate this mitigation, certificate authorities **SHOULD NOT** issue certificates with the policy mappings extension (Section 4.2.1.5 of [RFC5280]). Applications maintaining policies for accepted trust anchors are **RECOMMENDED** to forbid this extension in participating certificate authorities.

### 6.5. Disable Policy Checking

An X.509 validator can mitigate this attack by disabling policy validation entirely. This may be viable for applications that do not require policy validation. In this case, critical policy-related extensions, notably the policy constraints extension (Section 4.2.1.11 of [RFC5280]), **MUST** be treated as unrecognized extensions as described in Section 4.2 of [RFC5280] and be rejected.

## 7. Security Considerations

Section 3 discusses how the policy tree algorithm in [RFC5280] can lead to denial-of-service vulnerabilities in X.509-based applications, such as [CVE-2023-0464] and [CVE-2023-23524].

Section 5 replaces this algorithm to avoid this issue. As discussed in Section 4.1, the new structure scales linearly with the input. This means input limits in X.509 validators will more naturally bound processing time, thus avoiding these vulnerabilities.

## 8. IANA Considerations

This document has no IANA actions.

## 9. References

### 9.1. Normative References

**[RFC2119]** Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

**[RFC5280]** Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

**[RFC8174]** Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

## 9.2. Informative References

**[CVE-2023-0464]** CVE, "Excessive Resource Usage Verifying X.509 Policy Constraints", CVE-2023-0464, March 2023, <https://www.cve.org/CVERecord?id=CVE-2023-0464>.

**[CVE-2023-23524]** CVE, "Processing a maliciously crafted certificate may lead to a denial-of-service", CVE-2023-23524, February 2023, <https://www.cve.org/CVERecord?id=CVE-2023-23524>.

**[RFC8446]** Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

**[X.509]** ITU-T, "Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, October 2019, <https://www.itu.int/rec/T-REC-X.509>.

## Acknowledgements

The author thanks Bob Beck, Adam Langley, Matt Mueller, and Ryan Sleevi for many valuable discussions that led to discovering this issue, understanding it, and developing the mitigation. The author also thanks Martin Thomson, Job Snijders, and John Scudder for their review and feedback on this document.

## Author's Address

**David Benjamin**
Google LLC
Email: davidben@google.com