Workgroup: Network
Internet-Draft:
draft-pwouters-ipsecme-multi-sa-performance-02
Published: 13 October 2021
Intended Status: Standards Track
Expires: 16 April 2022

Authors: A. Antony T. Brunner S. Klassert P. Wouters
secunet codelabs secunet Aiven

IKEv2 support for per-queue Child SAs

Abstract

This document defines three Notify Message Type Payloads for the Internet Key Exchange Protocol Version 2 (IKEv2) indicating support for the negotiation of multiple identical Child SAs to optimize performance.

The CPU_QUEUES notification indicates support for multiple queues or CPUs. The CPU_QUEUE_INFO notification is used to confirm and optionally convey information about the specific queue. The TS_MAX_QUEUE notify conveys that the peer is unwilling to create more additional Child SAs for this particular Traffic Selector set.

Using multiple identical Child SAs has the benefit that each stream has its own Sequence Number Counter, ensuring that CPUs don't have to synchronize their crypto state or disable their packet replay protection.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

IPsec implementations are currently limited to using one queue or CPU per Child SA. The result is that a machine with many queues/CPUs is limited to only using one of these per Child SA. This severely limits the throughput that can be attained. An unencrypted link of 10Gbps or more is commonly reduced to 2-5Gbps when IPsec is used to encrypt the link using AES-GCM. By using the implementation specified in this document, aggregate throughput increased from 5Gbps using 1 CPU to 40-60 Gbps using 25-30 CPUs

While this could be (partially) mitigated by setting up multiple narrowed Child SAs, for example using Populate From Packet (PFP) as specified in [RFC4301], this IPsec feature is not widely implemented. Some route based IPsec implementations might be able to implement this with specific rules into separate network interfaces, but these methods might not be available for policy based IPsec implementations.

To make better use of multiple network queues and CPUs, it can be beneficial to negotiate and install multiple identical Child SAs. IKEv2 [RFC7296] already allows installing multiple identical Child SAs, it offers no method to negotiate the number of Child SAs or indicate the purpose for the multiple Child SAs that are requested.

When two IKEv2 peers want to negotiate multiple Child SAs, it is useful to be able to convey how many Child SAs are required for optimized traffic. This avoids triggering CREATE_CHILD_SA exchanges that will only be rejected by the peer.

1.1. Requirements Language

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.

2. Performance bottlenecks

Currently, most IPsec implementations are limited by using one CPU or network queue per Child SA. There are a number of practical reasons for this, but a key limitation is that sharing the crypto state, counters and sequence numbers between multiple CPUs is not feasible without a significant performance penalty. There is a need to negotiate and establish multiple Child SAs with identical TSi/TSr on a per-queue or per-CPU basis.

3. Negotiation of CPU specific Child SAs

When negotiating CPU specific Child SAs, the first SA negotiated either in an IKE_AUTH exchange or CREATE_CHILD_SA is called Fallback SA. This Child SA is similar to a regular Child SA in that it is not bound to a single CPU. This Fallback Child SA (or its rekeyed successors) MUST remain active for the lifetime of the IPsec session to ensure that there is always a Child SA that can be selected to send traffic over, in case a per-resource Child SA is not available. Additional Child SAs are installed bound to a specific CPU. These Child SAs are responsible for the bulk of the traffic.

The CPU_QUEUES notification payload is sent in the IKE_AUTH or CREATE_CHILD_SA Exchange indicating the negotiated Child SA is a Fallback SA.

The CPU_QUEUES notification value refers to the number of additional resource-specific Child SAs that may be installed for this particular TSi/TSr combination excluding the Fallback Child SA. Both peers send the preferred minimum number of additional Child SAs to install. Both peers pick the maximum of the two numbers (within reason). That is, if the initiator prefers 16 and the responder prefers 48, then the number negotiated is 48. The responder may at any time reject additional Child SAs by returning TS_MAX_QUEUE. It should not return NO_ADDITIONAL_SAS, as there might be another Child SAs with different Traffic Selectors that would still be allowed by the peer.

CPU-specific Child SAs are negotiated as regular Child SAs using the CREATE_CHILD_SA exchange and are identified by a CPU_QUEUE_INFO notification. Upon installation, each Child SA is associated with an additional local selector, such as CPU or queue. These additional Child SAs MUST be negotiated with identical Child SA properties that were negotiated for the Fallback SA. This includes cryptographic algorithms, Traffic Selectors, Mode (e.g. transport mode), compression usage, etc. However, the Child SAs do have their own individual keying material that is derived according to the regular IKEv2 process. The CPU_QUEUE_INFO can be empty or contain some identifying data that could be useful for debugging purposes.

Additional Child SAs can be started on-demand or can be started all at once. Peers may also delete specific per-resource Child SAs if they deem the associated resource to be idle. The Fallback SA MUST NOT be deleted while any per-resource Child SAs are still present.

During the CREATE_CHILD_SA rekey for the Child SA, the CPU_QUEUE_INFO notification MAY be included, but regardless of whether or not it is included, the rekeyed Child SA MUST be bound to the same resource(s) as the Child SA that is being rekeyed.

As with regular Child SA rekeying, the new Child SA may not be different from the rekeyed Child SA with respect to cryptographic algorithms and MUST cover the original Traffic Selector ranges.

If a CREATE_CHILD_SA exchange request containing both a CPU_QUEUE_INFO and a CPU_QUEUES notification is received, the responder MUST ignore the CPU_QUEUE_INFO payload. If a CREATE_CHILD_SA exchange reply is received with both CPU_QUEUE_INFO and CPU_QUEUES notifications, the initiator MUST ignore the notification that it did not send in the request.

The CPU_QUEUES notification, even when it is sent in the IKE_AUTH exchange, is not an attribute of the IKE peer. It is an attribute of the Child SA, similar to the USE_TRANSPORT notification. That is, an IKE peer can have multiple Child SAs covering different traffic selectors and selectively decide whether or not to enable additional per-resource Child SAs for each of these Child SAs covering different Traffic Selectors.

4. Implementation Considerations

There are various considerations that an implementation can use to determine the best way to install multiple Child SAs. Below are examples of such strategies.

A simple distribution could be to install one additional Child SA on each CPU. The Fallback Child SA ensures that any CPU generating traffic to be encrypted has an available (if not optimal) Child SA to use. Any subsequent Child SAs with identical TSi/TSr Traffic Selectors are installed in such a way to only be used by a single CPU or network queue.

Performing per-CPU Child SA negotiations can result in both peers initiating additional Child SAs at once. This is especially likely if per-CPU Child SAs are triggered by individual SADB_ACQUIRE [RFC2367] messages. Responders should install the additional Child SA on a CPU with the least amount of additional Child SAs for this TSi/TSr pair. It should count outstanding SADB_ACQUIREs as an assigned additional Child SA. It is still possible that when the peers only have one slot left to assign, that both peers send a CREATE_CHILD_SA request at the same time.

As an optimization, additional Child SAs that see little traffic MAY be deleted. The Fallback Child SA MUST NOT be deleted when idle, as it is likely to be idle if enough per-CPU Child SAs are installed. However, if one of those per-CPU child SAs is deleted because it was idle, and subsequently that CPU starts to generate traffic again, that traffic does not have a per-CPU Child SA and will be encrypted using the Fallback Child SA. Meanwhile, the IKE daemon might be negotiating to bring up a new per-CPU Child SA.

When the number of queues or CPUs are different between the peers, the peer with the least amount of queues or CPUs MAY decide to not install a second outbound Child SA for the same resource as it will never use it to send traffic. However, it MUST install all inbound Child SAs as it has committed to receiving traffic on these negotiated Child SAs.

If per-CPU SADB_ACQUIRE messages are implemented (see Section 6), the Traffic Selector (TSi) entry containing the information of the trigger packet should still be included in the TS set. This information MAY be used by the peer to select the most optimal target CPU to install the additional Child SA on. For example, if the trigger packet was for a TCP destination to port 25 (SMTP), it might be able to install the Child SA on the CPU that is also running the mail server process. Trigger packet Traffic Selectors are documented in [RFC7296] Section 2.9.

As per RFC 7296, rekeying a Child SA SHOULD use the same (or wider) Traffic Selectors to ensure that the new Child SA covers everything that the rekeyed Child SA covers. This includes Traffic Selectors negotiated via Configuration Payloads (CP) such as INTERNAL_IP4_ADDRESS which may use the original wide TS set or use the narrowed TS set.

5. Payload Format

All multi-octet fields representing integers are laid out in big endian order (also known as "most significant byte first", or "network byte order").

5.1. CPU_QUEUES Notify Status Message Payload

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-----------------------------+-------------------------------+
! Next Payload  !C!  RESERVED   !         Payload Length        !
+---------------+---------------+-------------------------------+
!  Protocol ID  !   SPI Size    !      Notify Message Type      !
+---------------+---------------+-------------------------------+
!  Minimum number of IPsec SAs                                  !
+-------------------------------+-------------------------------+
```

*Protocol ID (1 octet) - MUST be 0. MUST be ignored if not 0.

*SPI Size (1 octet) - MUST be 0. MUST be ignored if not 0.

*Notify Status Message Type (2 octets) - set to [TBD1]

*Minimum number of per-CPU IPsec SAs (4 octets). MUST be greater than 0. If 0 is received, it MUST be interpreted as 1.

Note: The Fallback Child SA that is not bound to a single CPU is not counted as part of these numbers.

5.2. CPU_QUEUE_INFO Notify Status Message Payload

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-----------------------------+-------------------------------+
! Next Payload  !C!  RESERVED   !         Payload Length        !
+---------------+---------------+-------------------------------+
!  Protocol ID  !   SPI Size    !      Notify Message Type      !
+---------------+---------------+-------------------------------+
!                                                               !
~               Optional queue identifier                       ~
!                                                               !
+-------------------------------+-------------------------------+
```

*Protocol ID (1 octet) - MUST be 0. MUST be ignored if not 0.

*SPI Size (1 octet) - MUST be 0. MUST be ignored if not 0.

*Notify Status Message Type (2 octets) - set to [TBD2]

*Optional Payload Data. This value MAY be set to convey the local identity of the queue. The value SHOULD be a unique identifier and the peer SHOULD only use it for debugging purposes.

5.3. TS_MAX_QUEUE Notify Error Message Payload

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-----------------------------+-------------------------------+
! Next Payload  !C!  RESERVED   !         Payload Length        !
+---------------+---------------+-------------------------------+
!  Protocol ID  !   SPI Size    !      Notify Message Type      !
+---------------+---------------+-------------------------------+
```

*Protocol ID (1 octet) - MUST be 0. MUST be ignored if not 0.

*SPI Size (1 octet) - MUST be 0. MUST be ignored if not 0.

*Notify Error Message Type (2 octets) - set to [TBD3]

*Optional Payload Data. Must be 0.

6. Operational Considerations

Implementations supporting per-CPU SAs SHOULD extend their local SPD selector, and the mechanism of on-demand negotiation that is triggered by traffic to include a CPU (or queue) identifier in their SADB_ACQUIRE message from the SPD to the IKE daemon. If the IKEv2 extension defined in this document is negotiated with the peer, a node which does not support receiving per-CPU SADB_ACQUIRE messages

MAY initiate all its Child SAs immediately upon receiving the (only) SADB_ACQUIRE it will receive from the IPsec stack. Such implementations also need to be careful when receiving a Delete Notify request for a per-CPU Child SA, as it has no method to detect when it should bring up such a per-CPU Child SA again later. And bringing the deleted per-CPU Child SA up again immediately after receiving the Delete Notify might cause an infinite loop between the peers. Another issue of not bringing up all its per-CPU Child SAs is that if the peer acts similarly, the two peers might end up with only the Fallback SA without ever activating any per-CPU Child SAs. It is there for RECOMMENDED to implement per-CPU SADB_ACQUIRE messages.

The minimum number of Child SAs negotiated should not be treated as the maximum number of allowed Child SAs. Peers SHOULD be lenient with this number to account for corner cases. For example, during Child SA rekeying, there might be a large number of additional Child SAs created before the old Child SAs are torn down. Similarly, when using on-demand Child SAs, both ends could trigger multiple Child SA requests as the initial packet causing the Child SA negotiation might have been transported to the peer via the Fallback SA where its reply packet might also trigger an on-demand Child SA negotiation to start. A peer may want to allow up to double the negotiated minimum number of Child SAs, and rely on idleness of Child SAs to tear down any unused Child SAs gradually to to reach an optimal number of Child SAs. Adding too many SAs may slow down per-packet SAD lookup.

Implementations might support dynamically moving a per-CPU Child SAs from one CPU to another CPU. If this method is supported, implementations must be careful to move both the inbound and outbound SAs. If the IPsec endpoint is a gateway, it can move the inbound SA and outbound SA independently from each other. It is likely that for a gateway, IPsec traffic would be asymmetric. If the IPsec endpoint is the same host responsible for generating the traffic, the inbound and outbound SAs SHOULD remain as a pair on the same CPU. If a host previously skipped installing an outbound SA because it would be an unused duplicate outbound SA, it will have to create and add the previously skipped outbound SA to the SAD with the new CPU ID. The inbound SA may not have CPU ID in the SAD. Adding the outbound SA to the SAD requires access to the key material, whereas for updating the CPU selector on an existing outbound SAs. access to key material might not be needed. To support this, the IKE software might have to hold on to the key material longer than it normally would, as it might actively attempt to destroy key material from memory that it no longer needs access to.

7. Security Considerations

[TO DO]

8. Implementation Status

[Note to RFC Editor: Please remove this section and the reference to [RFC6982] before publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC7942], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Authors are requested to add a note to the RFC Editor at the top of this section, advising the Editor to remove the entire section before publication, as well as the reference to [RFC7942].

8.1. Linux XFRM

Organization: Linux kernel XFRM

Name: XFRM-PCPU-v1 https://git.kernel.org/pub/scm/linux/kernel/git/klassert/linux-stk.git/log/?h=xfrm-pcpu-v1

Description: An initial Kernel IPsec implementation of the per-CPU method.

Level of maturity: Alpha

Coverage: Implements Fallback Child SA and per-CPU Child SAs. It only supports the NETLINK API. The PFKEYv2 API is not supported.

Licensing: GPLv2

Implementation experience: The Linux XFRM implementation added two additional attributes to support per-CPU SAs. There is a new attribute XFRMA_SA_PCPU, u32, for the SAD entry. This attribute should present on the outgoing SA, per-CPU Child SAs, starting from 0. This attribute MUST NOT be present on the Fallback XFRM SA. It is used by the kernel only for the outgoing traffic, (clear to encrypted). The incoming SAs, both the Fallback and the per-CPU SA, do not need XFRMA_SA_PCPU attribute. XFRM stack can not use CPU id on the incoming SA. The kernel internally sets the value to 0xFFFFFF for the incoming SA and the Fallback SA. However, one may add XFRMA_SA_PCPU to the incoming per-CPU SA to steer the ESP flow, to a specific Q or CPU e.g ethtool ntuple configuration. The SPD entry has new flag XFRM_POLICY_CPU_ACQUIRE. It should be set only on the "out" policy. The flag should be disabled when the policy is a trap policy, without SPD entries. After a successful negotiation of CPU_QUEUES, while adding the Fallback Child SA, the SPD entry can be updated with the XFRM_POLICY_CPU_ACQUIRE flag. When XFRM_POLICY_CPU_ACQUIRE is set, the XFRM_MSG_ACQUIRE generated will include the XFRMA_SA_PCPU attribute.

Contact: Steffen Klassert steffen.klassert@secunet.com

8.2. Libreswan

Organization: The Libreswan Project

Name: pcpu-3 https://libreswan.org/wiki/XFRM_pCPU

Description: An initial IKE implementation of the per-CPU method.

Level of maturity: Alpha

Coverage: implements Fallback Child SA and per-CPU additional Child SAs

Licensing: GPLv2

Implementation experience: TBD

Contact: Libreswan Development: swan-dev@libreswan.org

8.3. strongSwan

Organization: The StrongSwan Project

Name: StrongSwan https://github.com/strongswan/strongswan/tree/per-cpu-sas-poc/

Description:
An initial IKE implementation of the per-CPU method.

Level of maturity: Alpha

Coverage: implements Fallback Child SA and per-CPU additional Child SAs

Licensing: GPLv2

Implementation experience: StrongSwan use private space values for notifications CPU_QUEUES (40970) and QUEUE_INFO (40971).

Contact: Tobias Brunner tobias@strongswan.org

8.4. iproute2

Organization: The iproute2 Project

Name: iproute2 https://github.com/antonyantony/iproute2/tree/pcpu-v1

Description: Implemented the per-CPU attributes for the "ip xfrm" command.

Level of maturity: Alpha

Licensing: GPLv2

Implementation experience: TBD

Contact: Antony Antony antony.antony@secunet.com

9. IANA Considerations

This document defines two new IKEv2 Notify Message Type payloads for the IANA "IKEv2 Notify Message Types - Status Types" registry.

Value	Notify Type Messages - Status Types	Reference
[TBD1]	CPU_QUEUES	[this document]
[TBD2]	CPU_QUEUE_INFO	[this document]

Figure 1

This document defines one new IKEv2 Notify Message Type payloads for the IANA "IKEv2 Notify Message Types - Error Types" registry.

Value	Notify Type Messages - Status Types	Reference
[TBD3]	TS_MAX_QUEUE	[this document]

Figure 2

10. References

10.1. 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>.

[RFC2367] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, DOI 10.17487/RFC2367, July 1998, <https://www.rfc-editor.org/info/rfc2367>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <https://www.rfc-editor.org/info/rfc7296>.

[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>.

10.2. Informative References

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <https://www.rfc-editor.org/info/rfc6982>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

Authors' Addresses

Antony Antony
secunet Security Networks AG

Email: antony.antony@secunet.com

Tobias Brunner
codelabs GmbH

Email: tobias@codelabs.ch

Steffen Klassert
secunet Security Networks AG

Email: steffen.klassert@secunet.com

Paul Wouters
Aiven

Email: paul.wouters@aiven.io