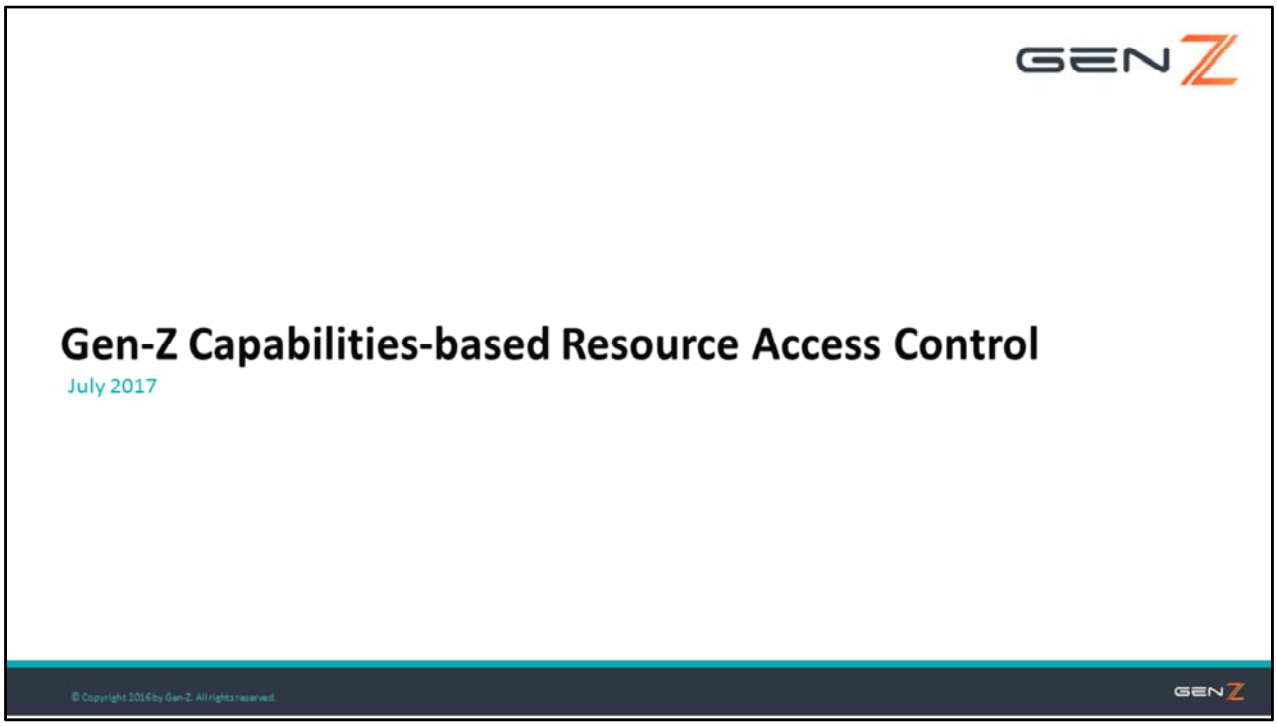


This presentation covers capabilities-based resource access control when performing read and write operations. These operations complement the research work described in: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-850.pdf

Disclaimer

This document is provided 'as is' with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. Gen-Z Consortium disclaims all liability for infringement of proprietary rights, relating to use of information in this document. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Gen-Z is a trademark or registered trademark of the Gen-Z Consortium.

All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.

All material is subject to change at any time at the discretion of the Gen-Z Consortium

http://genzconsortium.org/

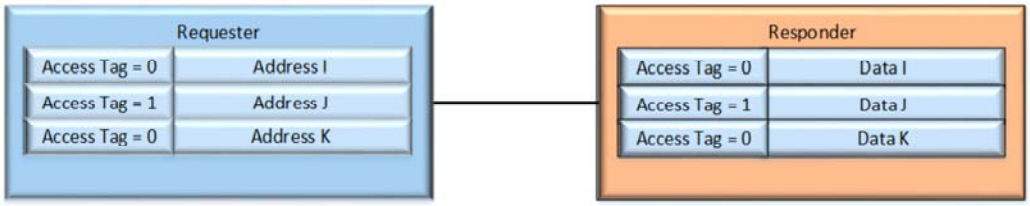
Problem

- Many security defects and vulnerabilities stem from referencing pointers that are outside the bounds of data buffers.
  - Critical system software is often written in assembly and/or low-level programming languages which lack pointer bounds checking.
  - Moving to a high-level language is a potential solution but is often not practical for reasons such as performance.
  - 'Fat pointers', which associate bounds with every pointer and check these bounds on every memory access, can enforce spatial safety and eliminate these violations.
    - However, fat pointers are not extensively used in practice because of overhead.

Capability-based Resource Access Control

- Capability-based addressing, which is also referred to as 'capabilities,' extends the fat pointer concept by encoding access rights in the form of 'handles' to memory.
  - A handle is a protected (i.e. unforgeable) object created only via the use of a privileged instruction or process such that a system can rely entirely on capabilities to manage memory.
  - Processes can share the same address space as each process can access only those objects it has capabilities to.
    - Processes can restrict only access rights in a capability but can never expand them.
      - Makes sharing objects as simple as giving each process a capability to the object.
- Capabilities can be implemented in software or hardware (figure illustrates hardware example)
  - Address is virtual address of the object
  - Offset is the extent of the object
  - GT is the Guard Tag is used to identify valid capabilities and to prevent forgeries
  - Permission is read-only or read-write

Ex. Requester and Responder with Capability Access Tag

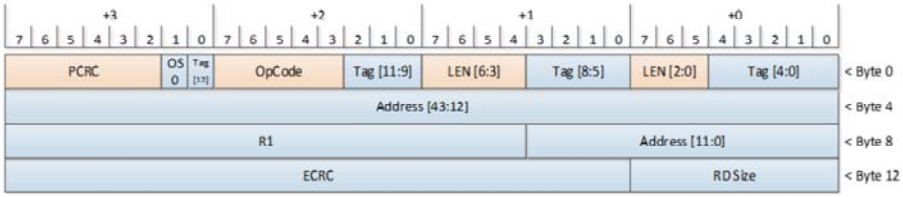


- This example uses a single-bit access tag where 1b indicates capability access control and 0b indicates no capability access control.
- In this example, a set of Requester addresses I-K have been configured such that address J requires capability access control and addresses I and K do not.
- Similarly, the Responder has been configured such that data I-K have been configured such that data J requires capability access control and data I and K do not.
- When the Responder receives a Capabilities Read or a Capabilities Write request packet, it examines the access tag associated with the data to determine what action to take.

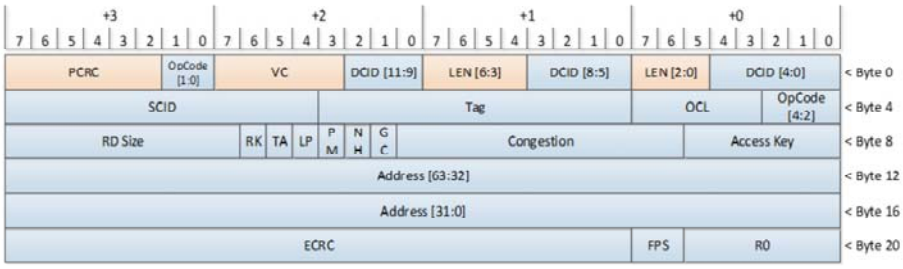
GEN Z

See specification for the detailed steps taken upon receipt of a Capabilities Read or Capabilities Write request packet.

Capabilities Read Request Packet Format



P2P-Core Capabilities Read Packet Format

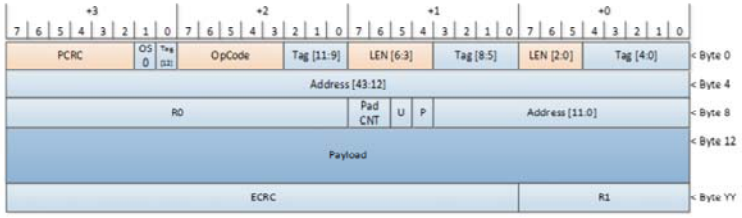


Core 64 Capabilities Read Packet Format

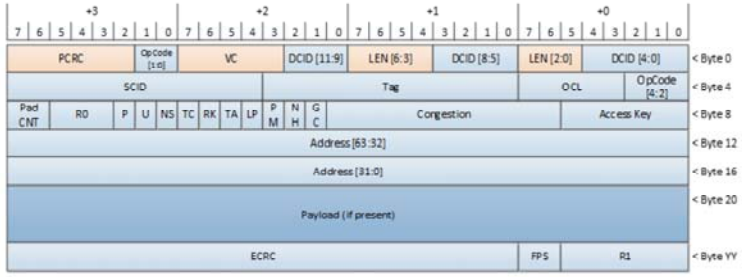
Core 64 Capabilities Read packet format is the same one used for Core 64 Read request packets (just a change in OpCode)

If a *Capabilities Read* request packet is received, then the Responder shall verify that an access tag is associated with the targeted resource and that the access tag indicates the targeted resource is capabilities data, e.g., is set to 1b if using a single-bit access tag. If the access tag is present and indicates the targeted resource is capability data, then the Responder shall execute the request packet and transmit a corresponding Read Response packet. If the access tag is not present or, if present, indicates the targeted resource is not capability data, then the Responder shall generate a *Standalone Acknowledgment* packet that indicates an Access Error (AE)—Invalid Capabilities Access.

Capabilities Write Request Packet Formats



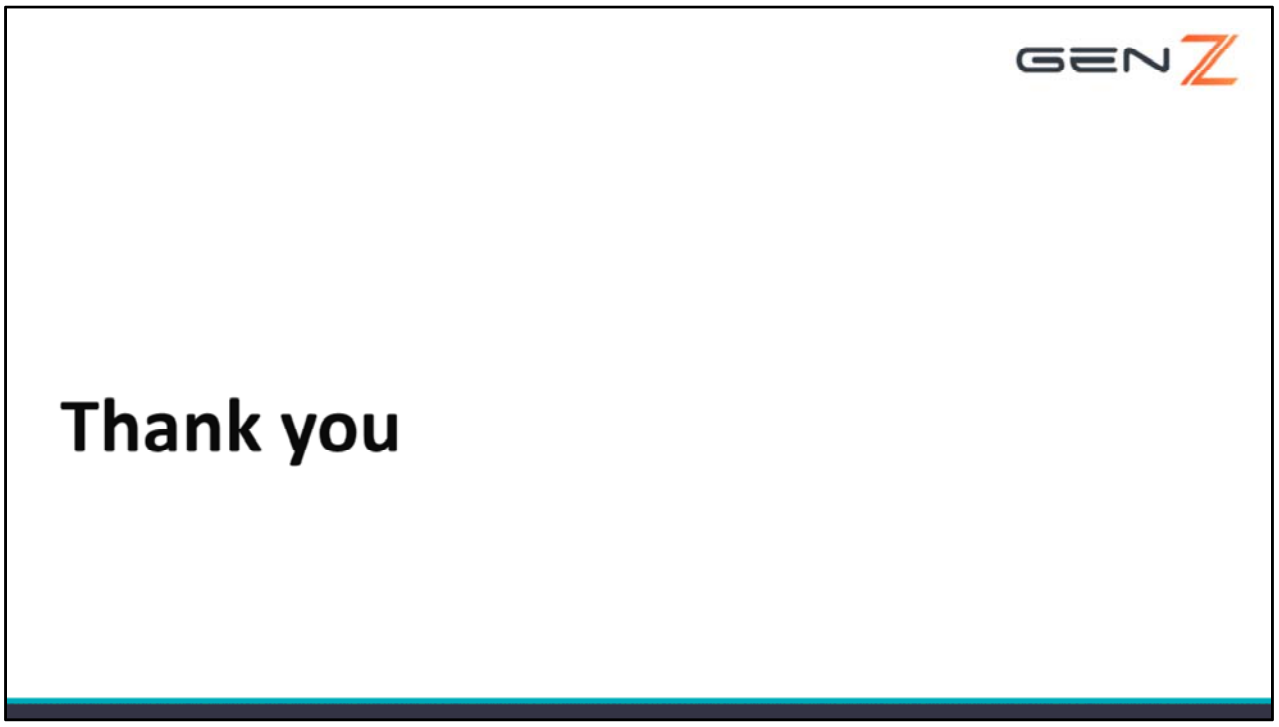
P2P-Core Capabilities Write Packet Format



Core 64 Capabilities Write Packet Format

Core 64 Capabilities Write packet format is the same one used for Core 64 Write request packets (just a change in OpCode)

If a *Capabilities Write* request packet is received, then the Responder shall verify that there is an access tag associated with the targeted resource. If the access tag is not present, then the Responder shall generate a *Standalone Acknowledgment* packet that indicates an Access Error (AE)—Invalid Capabilities Access. If the access tag is present, then the Responder shall execute the *Capabilities Write* request packet and update the access tag to indicate this resource contains capability data. The Responder shall ensure updates to the targeted resource and to the access tag are done atomically, i.e., execution of another request to the same resource is delayed until the updates to both are successfully completed.



This concludes this presentation. Thank you.