Hardware support for software security & emerging technology

COSC312 / COSC412

Learning objectives

- Appreciate that **hardware support** for software security is continuing to evolve
- Understand usefulness of emerging **capability models** in hardware and software
- Outline how **data flow security** can be effected using decentralised information flow control
- Explain the increasing need for **accountability**, including **data provenance**

Approximate ranking of vulnerabilities 1/2

- **Vulnerable dependencies**
  - Late application of software updates; zero-day attacks

- **Handling user input**
  - buffer-based attacks (*e.g.*, buffer overflow)
  - SQL injection
  - XSS—cross site scripting; *e.g.*, effects akin to session hijacking
  - Unauthorised directory traversal
  - DoS—*i.e.*, denial of service

Approximate ranking of vulnerabilities 2/2

- Failing to apply **principle of least privilege** (POLP)
  - APIs that are too open regarding permissions
  - Privilege escalation—lack of defence in depth
  - Insider threats / phishing—again, lack of defence in depth
- **System misconfiguration / design errors**
  - Missing application of encryption
  - Missing access control checks
  - Cloud specific risks: understanding shared security responsibility
    - *e.g.*, policy errors (open S3 buckets), vulnerability checking, *etc*.

Emerging CPU hardware-based security

- Intel SGX provides for ‘**enclaves**’ of secure software
  - RAM content only ever **decrypted when on CPU** die
  - First revision had limits on memory volume within enclaves
  - SGX upholds confidentiality, integrity, but **not availability**
  - Only for **user-space code**: no system call support
    - Involved in research ‘porting’ Docker to use SGX:
    - Get around userspace limitation by using a library operating system
    - Performance boosted by minimising enclave entry/exit transitions

- SGX is **coarse-grained protection**, unlike capabilities…

Hardware-supported memory capabilities

- Mentioned capabilities within software:
  - As a **collection of privileges** a particular principal holds
  - Taken into programming within the **E programming language**

- Capability machines add support within CPU hardware
  - All use of memory capabilities will be **checked for valid use**
    - *e.g.*, has effect of bounds-checking all pointer accesses
    - … where bounds-checks can be fine-grained, *e.g.*, byte granularity
  - **Cannot miss access control checks** (if no hardware bugs)

Cambridge CAP computer (1976)

- CAP had **hardware capabilities**:
    - capability unit with 64 cap. registers
- Each capability contains:
    - **Base** memory address
    - **Limit** to the range accessible
    - **Access permission** bits
        - R/W to memory; also R/W to capabilities
- Memory up to 64K 32-bit words
    - Access to memory required capability

COSC312 / COSC412 Lecture 12, 2024—CC BY-SA [image]() by [User:Daderot]() from Wikipedia

Cambridge CHERI development

- That's **Capability Hardware Enhanced RISC Instructions**
  - *i.e.*, a RISC CPU with an extended instruction set architecture

- CHERI is a **hybrid capability architecture**:
  - CHERI can run legacy code as well as 'pure' capability code
    - Pragmatic choice: pure-cap requires significant reengineering
    - Legacy code's memory access wrapped in default capabilities

- CHERI can be run in many ways:
  - *e.g.*, within the QEMU emulator; also on FPGA; and…

Arm Morello within UK DSbD programme

- **Morello** is a limited-supply evaluation system from Arm
  - (DSbD: Digital Security by Design, UK Gov.'s industry support)
  - Includes a modified Neoverse N1 system-on-chip (superscalar)
    - Arm's Neoverse CPU designs are their pitch for data centres
  - Aim is to allow **realistic testing of capability software**
- **CheriBSD** (cap-aware FreeBSD) is supported on Morello
  - … also efforts made to port of **Linux and Android**
- Meanwhile work also proceeding on **RISC-V support**

CHERI's (memory) capability design

- 64-bit pointers are changed to **128-bit capabilities**
  - Capabilities include: address; bounds; permissions
  - Address bounds are compressed—not byte-level granularity
  - Caps. **can be 'sealed'**: immutable; can't be dereferenced
- CHERI uses bitmap to mark **capabilities within RAM**
- **Strong security properties**:
  - Every CPU load, store and instruction fetch is cap-checked
  - Capabilities can only be derived from existing capabilities
    - Derived capabilities never increase range or permissions

Rate control

- Intelligence organisations apply **rate control protection**
    - *i.e.*, **avoid rapid exfiltration** of data by limiting coms. rates
    - Software answering queries may not need high bandwidth

- Not much use of rate control in today's OS software
    - … yet there are many cases of **large-scale exfiltration**!

- Fine-grained security hardware, *e.g.*, capabilities may provide the means to embed rate control effectively

Toward more general data flow security

- Event-based software: **message passing & processing**
  - Near real-time analysis versus store & query RDBMS approach

- Background: events versus messages?
  - **Messages** use stateful protocols; explicit routing information
  - **Events** are self-contained; amenable to content-based routing

- Off the shelf software: **distributed stream processing**
  - automated trading, business activity monitoring, sensor nets.

Distributed access control

- Recall approx. partition of access control schemes:
  - **Discretionary** (DAC): owners of resources control privileges
  - **Mandatory** (MAC): system-wide, consistent rules are applied

- Distribution adds extra challenges:
  - **Revocation** of privileges
  - **Disseminating** and **evolving** access control policy
  - (Explored some of these challenges within our 'OASIS' distributed role-based access control system)

Information flow control (IFC)

- IFC is a form of data-centric mandatory access control
  - Uses **security labels**: classified, secret, top secret, ...
  - All data items are labelled
  - All security principals operate at a labelled level
  - Simple **limiting rules applied consistently**: *e.g.*, “no write down”

- Appealing for end-to-end security properties:
  - Can manage both confidentiality and integrity checking
  - Closely related to taint propagation models

DIFC and event-based systems

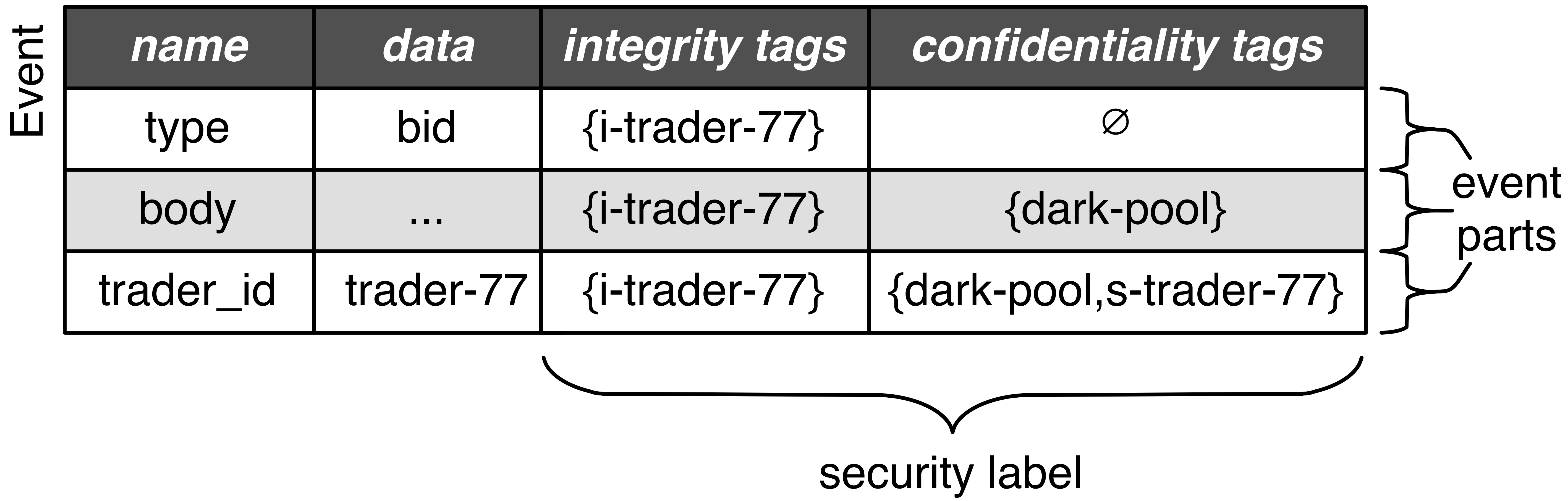
- **Decentralised IFC**: security label set is managed dynamically
  - Principals can create new labels, and issue privileges over them

- (D)IFC has been applied in **programming languages** & **OSs**
  - e.g., Asbestos (UCLA), Flume (MIT), JIF (Cornell), D-star (Stanford)

- However event-based systems are also highly compatible with (decentralised) information flow control techniques
- **Decentralised Event Flow Control** (DEFC):
  - DEFC is a labelling model that works below the granularity of events

(D)IFC in event-based systems

- Can treat all messages as **multi-part structures**
  - Apply IFC labelling independently to each part
  - Each part has its own **data and security label**
  - Middleware enforces data-linked access control to event parts



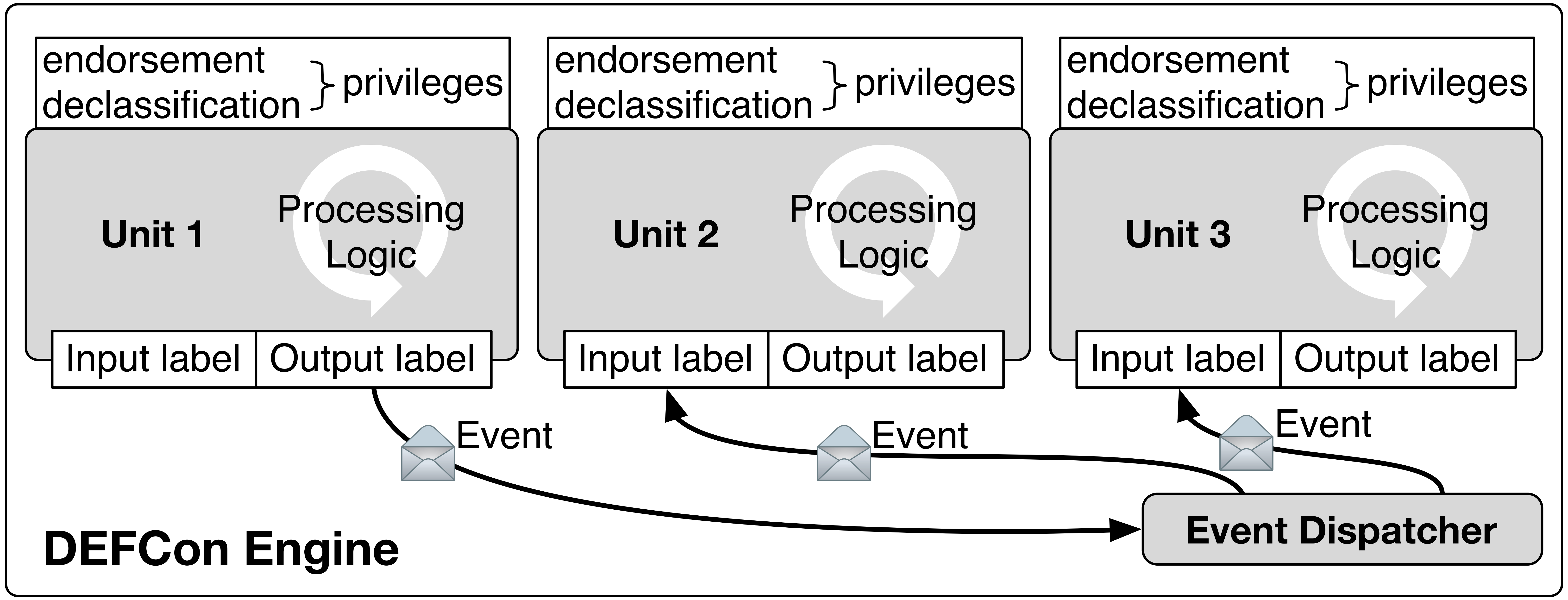
name	*data*	*integrity tags*	*confidentiality tags*
type	bid	{i-trader-77}	∅
body	...	{i-trader-77}	{dark-pool}
trader_id	trader-77	{i-trader-77}	{dark-pool,s-trader-77}

Event
event parts
security label

- For data transport, an **event is an atomic unit**
  - External and internal event transport can use different technologies

DEFCon: a DEFC prototype

- Trusted engine **handles lifecycle** of event proc. units
  - Engine also manages the lifecycle of security labels
  - Tracks which units are operating with which label privileges
  - Tracks labelling of data



DEFCon case study: pairs trading

- **Pairs trading** is a simple algorithm for stock trading
  - Assume that stocks in a related market will move in a related way
  - Look for divergence and bet on divergence soon reducing
- Modern exchanges are electronic, event-based systems:
  - Orders are quick transactions, pub/sub for dissemination of stocks
- Competition in the market makes it very latency sensitive
  - Co-host trading algorithms on server co-located with exchange
- Large brokers can offer '**dark pools**': avoids exchange

DEFCon case study: pairs trading

- Used DIFC to support safely co-located pairs trading
  - Protection of **market data integrity**
  - Strict control of interactions between **clients' investment strategies**
  - Ensure **confidentiality of orders** in 'dark pool' trading
    - Bulk-move equities without revealing trader's identity
  - Allow **auditing by regulatory authorities** on completed transactions
- Treating stock-exchange hosting as a 'cloud' (in effect)
- DIFC protects traders' strategies while efficiently utilising infrastructure

Pairs trading event model

owns s
Stock Exchange
Tick
MSFT
1234$
12:32:13
Match
buy: GOOG
sell: MSFT
mean: 1.92
3
owns t_1
4
Bid
sell: MSFT
price: 1234$
name: Trader 1
owns b
5
Pair Monitor
secrecy: t_1
integrity: s
Monitor
symbol1: GOOG
symbol2: MSFT
mean: ...
sd: ...
1
Trader 1
Broker
Trade
price: 1234$
symbol: MSFT
buyer: Trader 1
seller: Trader 2
6
Tick
MSFT
1230$
12:22:13
2
owns t_2
Pair Monitor
secrecy: t_2
integrity: s
Monitor
symbol1: GOOG
symbol2: MSFT
mean: ...
sd: ...
Trader 2
7
Ask
buy: MSFT
price: 1234$
name: Trader 2
Delegation
user: Trader 2
KEY:
Unit that can declassify all of its state
Unit that cannot declassify any of its state
Unit that has some state that it cannot declassify
Event
Tagged by Trader 1
Tagged by Trader 2
Match
buy: MSFT
sell: GOOG
mean: 1.92
Warning
msg: Trading volume exceeded quota
8
owns r,s
Regulator
9

CamFlow—IFC in Linux kernel

- SmartFlow project—DIFC engine **confined to JVM**
  - Useful proof-of-concept, but can't work with legacy software

- CloudSafetyNet project—built **CamFlow DIFC engine**
  - Linux kernel implementation written by Thomas Pasquier
  - Freely available as open source via https://camflow.org

- CamFlow has an **explicit IFC API** for new codebases…
- … but it can also apply policy to **existing codebases**

Defining data provenance

- **Data provenance** examines:
  - where data came from;
  - what transformations were applied to it; and
  - what subsequent data now depends on this data

- Can help analyse any data handling, but in particular:
  - Used to ensure **reproducibility** in experimental workflows
  - Employed to checking **compliance** with policy (*e.g.*, privacy)
  - It's common to reconstruct provenance for ***post hoc* analyses**

DIFC for provenance tracking

- DIFC provides a means to **track data provenance**
  - Use labels to propagate provenance metadata dynamically
- Provenance can be **examined at runtime** giving:
  - (1) reduced log storage requirements; (2) responsive alerts
- Applied CamFlow IFC engine to provenance tracking
  - CamFlow already provides kernel and user-space functionality
  - Application-level semantics can guide provenance filtering

CamQuery is our provenance engine

- A CamFlow module that **effects provenance capture**
  - Efficient run-time graph queries
  - Can use W3C standard PROV representation

- Demonstrated its use **within the security domain**
  - Kernel-mode operation can apply policy rules proactively

- Code is available via the **CamFlow project on GitHub**
  - Including demos too, to ease your adoption of the code

Accountability of cloud computing

- Recent surges: cloud; AI/ML **unregulated** by law/state
  - … but have far-reaching effect: privacy, democracy, security

- EU General Data Protection Regulation
  - Finally, far-reaching effect in terms of **responsible computing**
  - Now the Digital (Services|Markets) Act (DSA & DMA)

- Dagstuhl Seminar 18181: Towards accountable systems
  - Brought together CS, law, and experts in public policy

Accountability Engineering

- Abstractions are needed to bring accountability into of computing—**accountability engineering**

- Want to add **accountability monitors** to software:
  - Monitor uses dynamic provenance to check application state
  - Focuses run-time feature extraction from software behaviour

- Decentralised accountability could use blockchain…
  - … but permissioned blockchains, preferably

In summary

- Discussed **vulnerabilities** considered most often abused
- Presented emerging **capability hardware**
- Outlined developments in **data flow security** including decentralised information flow control
- Described the **CamFlow provenance system**
- Highlighted that **accountability engineering** will be increasingly needed within future computing systems