Secure by Design — Ideas and Techniques

SecAppDev 2026

Daniel Deogun, Dan Bergh Johnsson

Leuven, June 2nd 2026

omega point.

About us

Daniel Deogun and Dan Bergh Johnsson are authors of the book Secure by Design. They are developers at heart and have been working with secure application development for 20+ years.

Both strongly believe security should be treated as a concern rather than a feature of its own. This mindset has helped them identify good design principles that results in secure software without adding extra complexity.

Dan and Daniel are established speakers that often present at international conferences on topics regarding high-quality development and security.

Daniel Deogun

Dan Bergh Johnsson

omega point.

Agenda

- Secure by Design as a method
- Domain Primitives as pattern for Integrity
- Applying Secure by Design in legacy code
- Patterns for Confidentiality
- Patterns for Traceability
- Secure by Design on an architectural level
- Advanced Persistent Threats vs 3 R’s of Enterprise Security

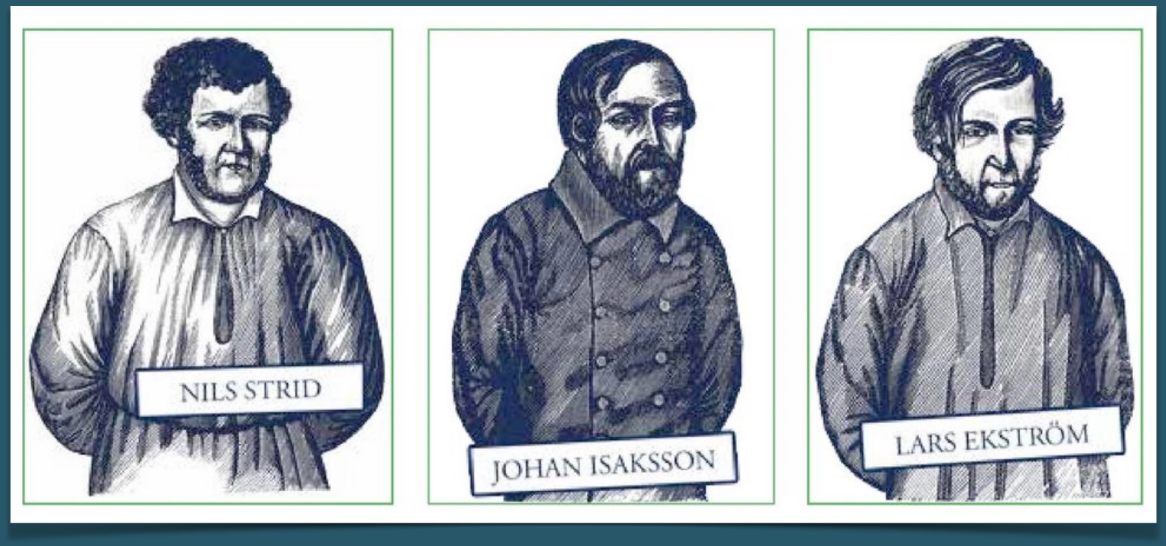
Primary Goal of this Session

- Gain insights about how higher security can spring out of a deliberate use of design
- See the connection between security concerns (CIA-T) and software design
- Be inspired to start securing your own code and architecture, and finding your own Secure by Design patterns

SECURE BY DESIGN

A Bank Robbery

- Öst-gotha Bank
- Linköping, Sweden 1854
- Largest bank heist in history at the date
- Several layers of high-class security features
- Advanced security procedures



SECURE BY DESIGN

But what about Digital Security?

”Somebody else’s problem”

90-day patching

Equifax

Meltdown

”Internetsladd i hårddisken”

Taken for granted

“you’re pwned!”

“Security is hard”

Security as a feature

SECURE BY DESIGN

CIA+ Triad

C **Confidentiality** – data needs to be protected from unauthorized access

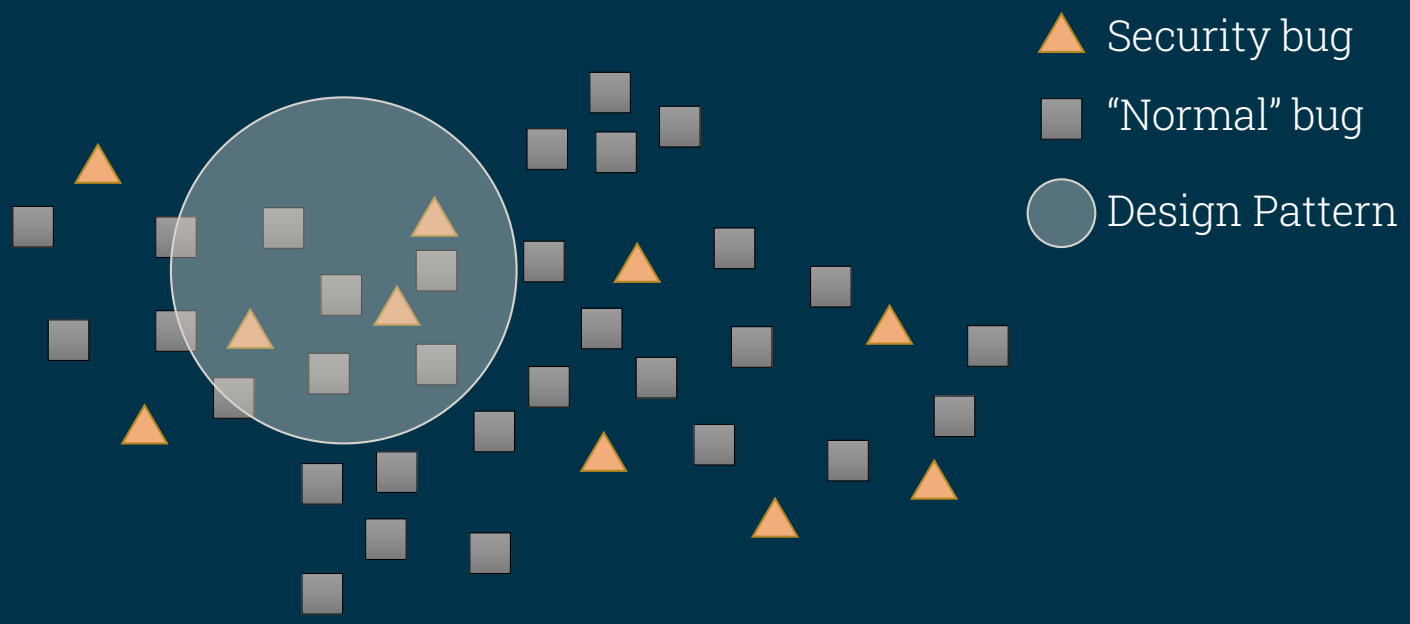
I **Integrity** – protect data from deletion or modification from an unauthorized party

A **Availability** – data must be available when needed

+

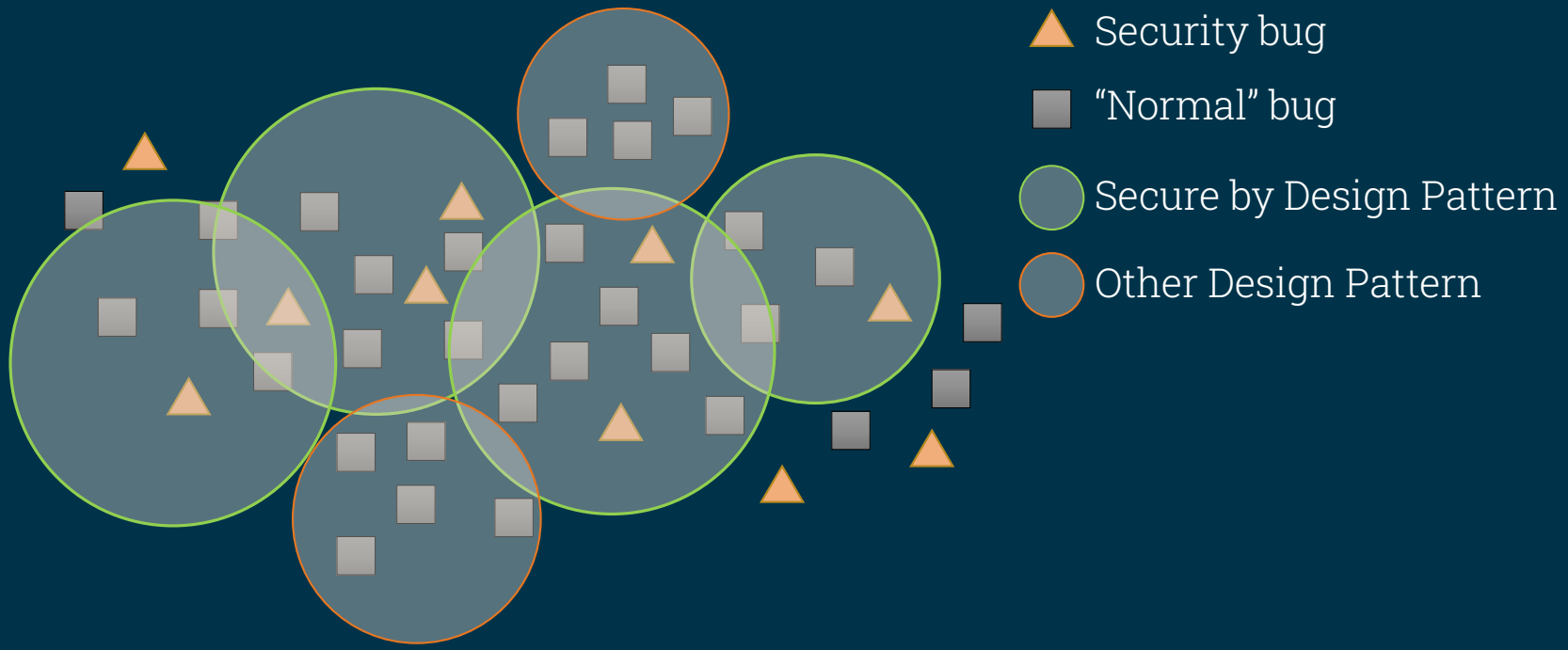
T **Traceability** (Non-repudiation, Auditability) – who changed what and when

Secure by Design – what's it all about?



SECURE BY DESIGN

Secure by Design – what's it all about?



Secure by Design

"A mindset and strategy for creating secure software by focusing on good design principles"

UK National Cyber Security Center

"Appropriate and proportionate cyber security measures are embedded within the delivery of digital services from the start and security posture is continually assured throughout the digital life cycle"

America's Cyber Defense Agency

"Secure by design means that technology products are built in a way that reasonably protects against malicious cyber actors successfully gaining access to devices, data, and connected infrastructure"

European Commission

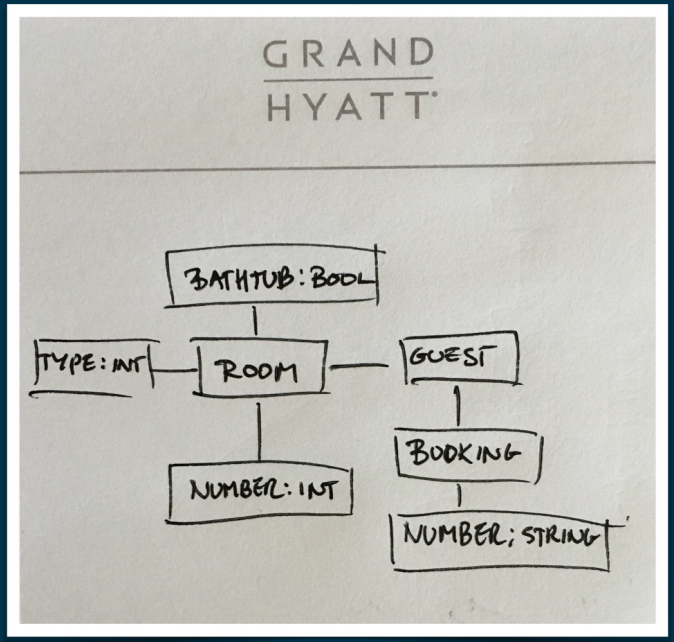
"Security by design (or secure by design), sometimes abbreviated "SbD," is a new industry term for a range of security practices built on one fundamental idea — that security should be built into a product by design, instead of being added on later by third-party products and services."

SECURE BY DESIGN

Integrity

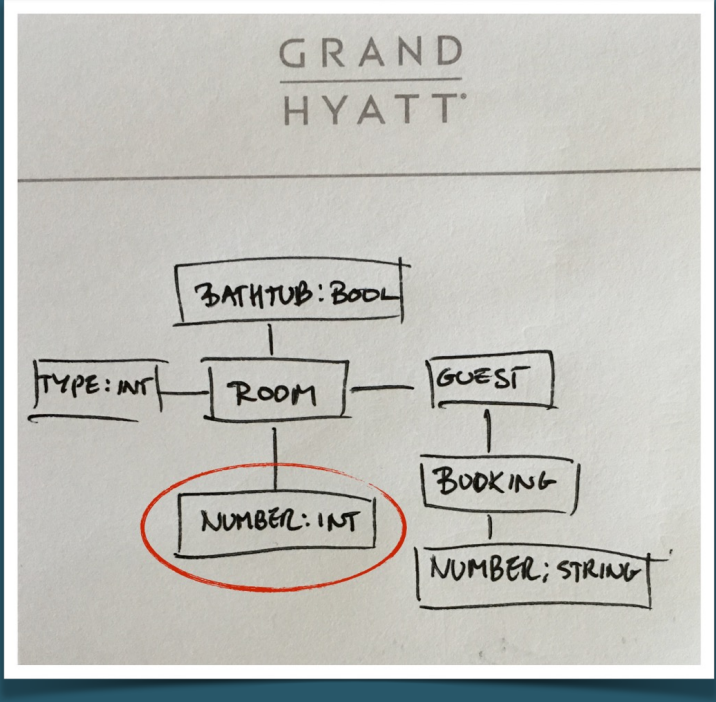
SECURE BY DESIGN

Let’s model a hotel room



SECURE BY DESIGN

But if a room number is just an integer



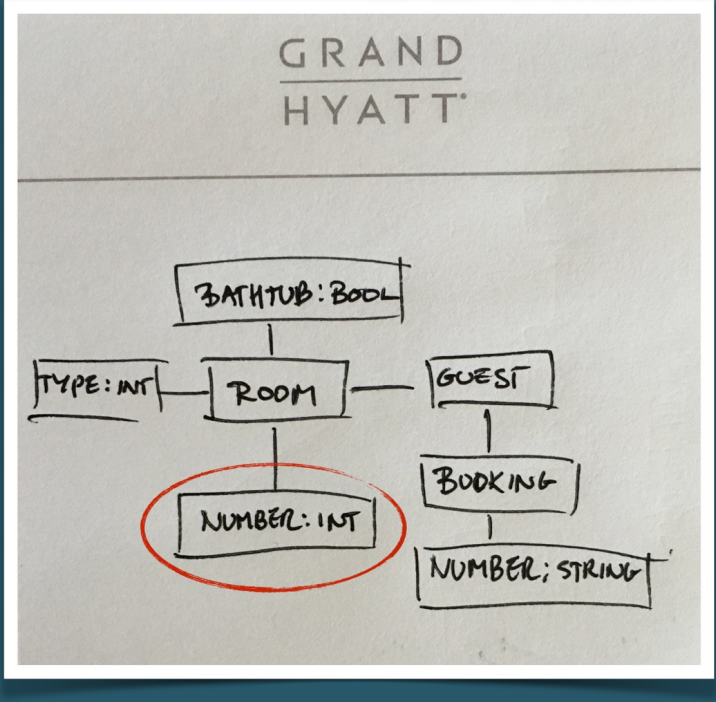
Peano's Axioms

- Zero is a number
- If n is a number, the successor of n is a number
- Zero isn't the successor of a number
- Two numbers of which the successors are equal are themselves equal
- If a set S of numbers contains zero and also the successor of every number in S, then every number is in S

Abelian Group

- Closure: a + b = integer
- Associativity: a + (b + c) = (a + b) + c
- Commutativity: a + b = b + a
- Identity: a + 0 = a
- Inverse: a + (−a) = 0

… or if we treat it as a String



Treating a room number as a String

- What characters are legal?
- Which operations can you apply?
- Does this really make sense?

Domain Primitives

"A value object so precise in its definition that it, by its mere existence, manifests its validity is called a Domain Primitive."

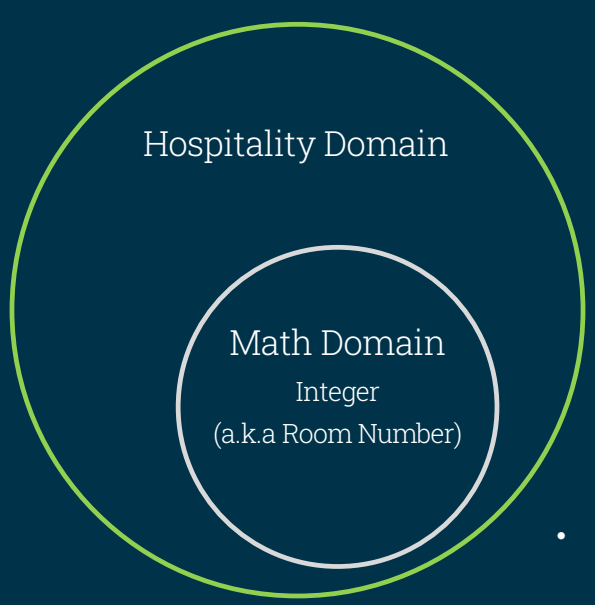
- Secure by Design

Interesting Properties

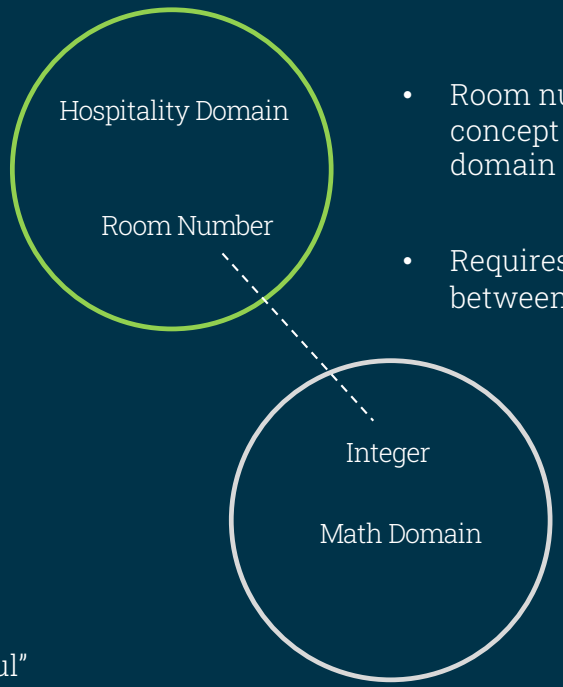
- Immutable and tightly coupled to the business domain
- Can only exist if and only if its value is valid in the context where it's used
- Reusable building block that's native to the domain and a conceptual whole
- Significantly reduces attack vectors to only semantically valid data.

SECURE BY DESIGN

Seen from a Contextual Perspective



- Integer is part of the Math domain
- An integer is far more “powerful” than a room number
- Representing a room number by an integer results in “unwanted” functionality



- Room number is a native concept in the Hospitality domain
- Requires explicit mapping between domains

SECURE BY DESIGN

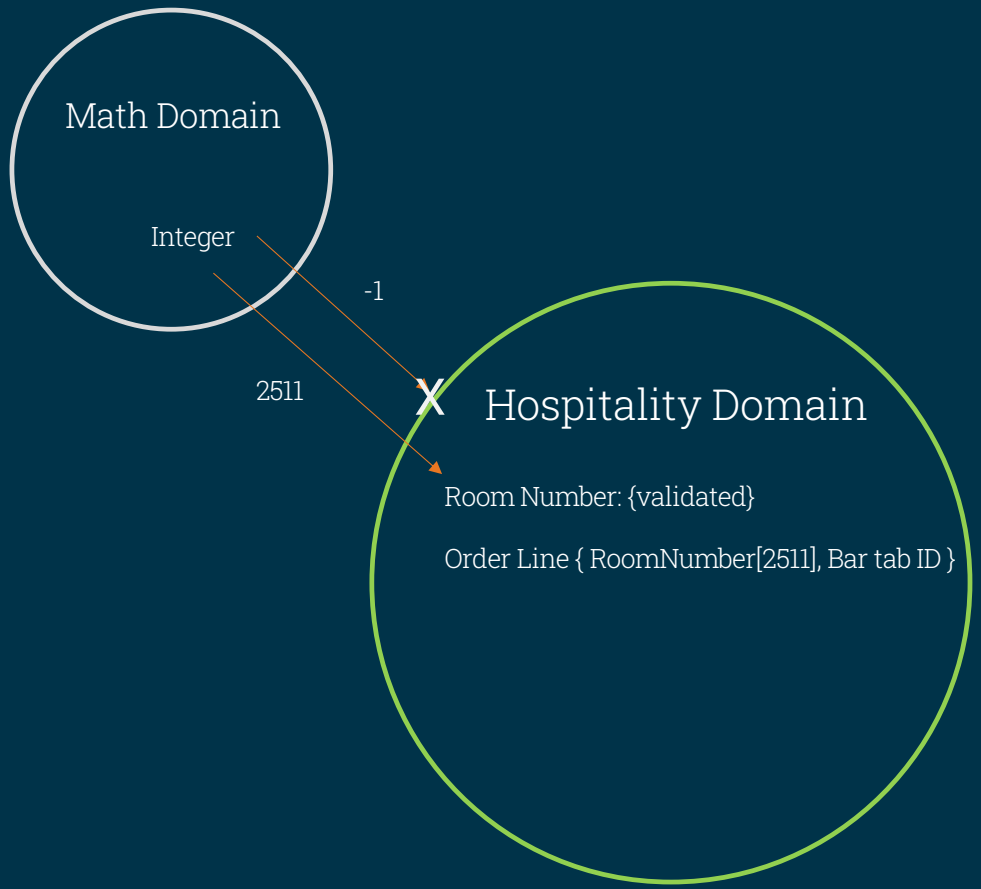
Room Number as a Domain Primitive

```
public final class RoomNumber {
   private final int value;

   public RoomNumber(final int value) {
      isTrue(Floor.isValid(value));
      inclusiveBetween(1, 50, value % 100);

      this.value = value;
   }

   public Floor floor() {
      return new Floor(value);
   }

   // other room number logic …
}
```

- Room Number is based on a language primitive “int” – is this, ok?
- Which floors are valid?
- How many rooms are available per floor?
- What does `//other room number logic …` mean?

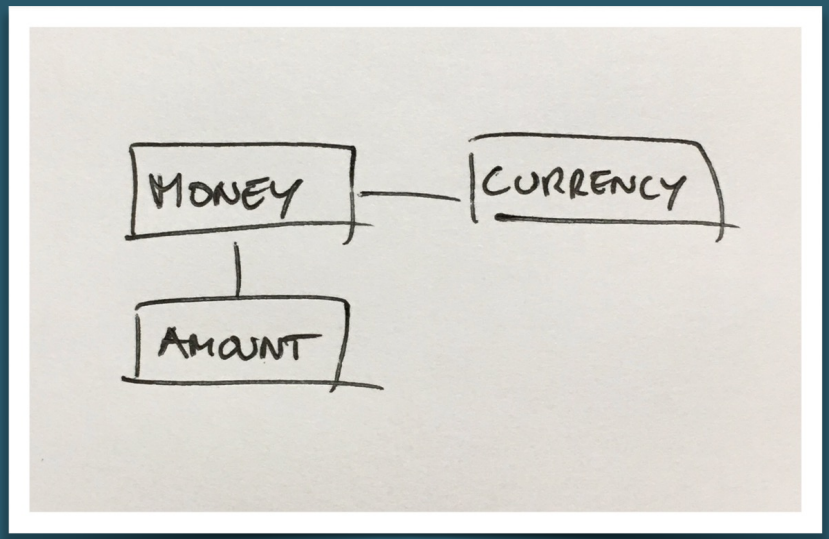
SECURE BY DESIGN

Seen from a Security Perspective



- Invalid room numbers are rejected by design
- Only valid room numbers are accepted
- This reduces attack vectors to only semantically valid data!

… but are Domain Primitives simple objects?




```
public void pay(final double money, final int recipientId) {
   final String currency = CurrencyService.currencyFor(recipientId);
   BankService.transfer(money, currency, recipientId);
}
```

But Money is a conceptual whole and should be modeled as a domain primitive

```
public void pay(final Money money, final Recipient recipient) {
   notNull(money);
   notNull(recipient);
   BankService.transfer(money, recipient);
}
```

SECURE BY DESIGN

Intelligent Machines - not just values


```
class Rate {
   private final Currency from;
   private final Currency to;

   Rate(Currency from, Currency to) {
      this.from = notNull(from);
      this.to = notNull(to);
   }

   Money exchange(Money from) {
      notNull(from);
      isTrue(this.from.equals(from.currency));
      BigDecimal converted = . . .
      return new Money(converted, to);
   }
}
```

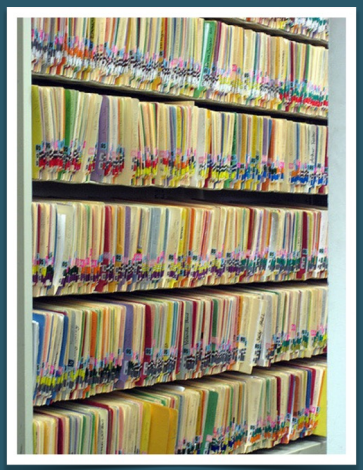
Standing on the Shoulders of Giants

- Domain Primitives act as building blocks
- Guy L. Steele Jr. “Growing a Language”
- Abelson, Sussman “Structure and Interpretation of Computer Programs”

SECURE BY DESIGN

But...

… it becomes a lot of classes!



https://flic.kr/p/2pvb2T

… isn't this overly complex?

https://flic.kr/p/VxQCpJ

… what about performance?

https://flic.kr/p/eGYhMw

Summary – Integrity

Integrity benefits from domain primitives by

- capturing business concepts, thus addressing data integrity
- ensuring valid data through a structured validation pattern
- counteracting injection attacks

SECURE BY DESIGN

Domain Primitives in Action on Legacy Code

https://flic.kr/p/O7zV

https://flic.kr/p/djYc9H

SECURE BY DESIGN

Three Steps Evolution Model

Draw the Line

https://flic.kr/p/nEZKMd

Harden your API

https://flic.kr/p/YgfS6s

Untangle Inside

https://flic.kr/p/wdBcT

Evolution Steps

Untangle Inside

Harden your API

Draw the Line

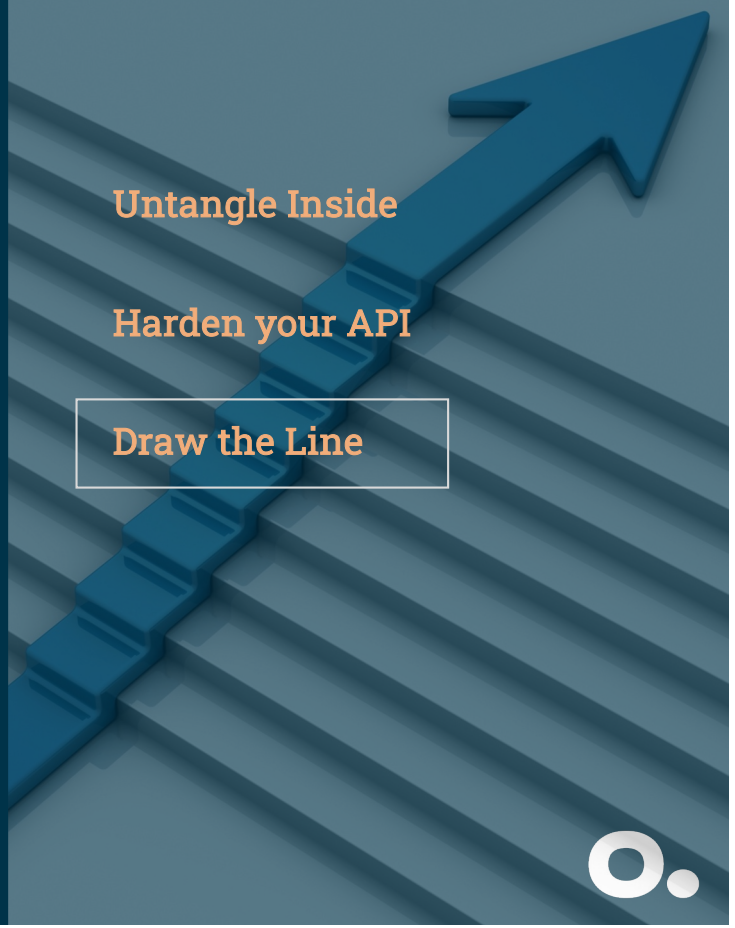
SECURE BY DESIGN

Establish Semantic Border – Observe Data

```
public void checkout(final int roomNumber) {
   houseKeepingService.registerForCleaning(roomNumber);
   minibarService.replenish(roomNumber);
   // other operations ...
}
```

```
public void checkout(final int roomNumber) {
   if (!RoomNumber.isValid(roomNumber)) {
      reporter.logInvalidRoomNumber(roomNumber);
   }

   houseKeepingService.registerForCleaning(roomNumber);
   minibarService.replenish(roomNumber);
   // other operations ...
}
```

Evolution Steps



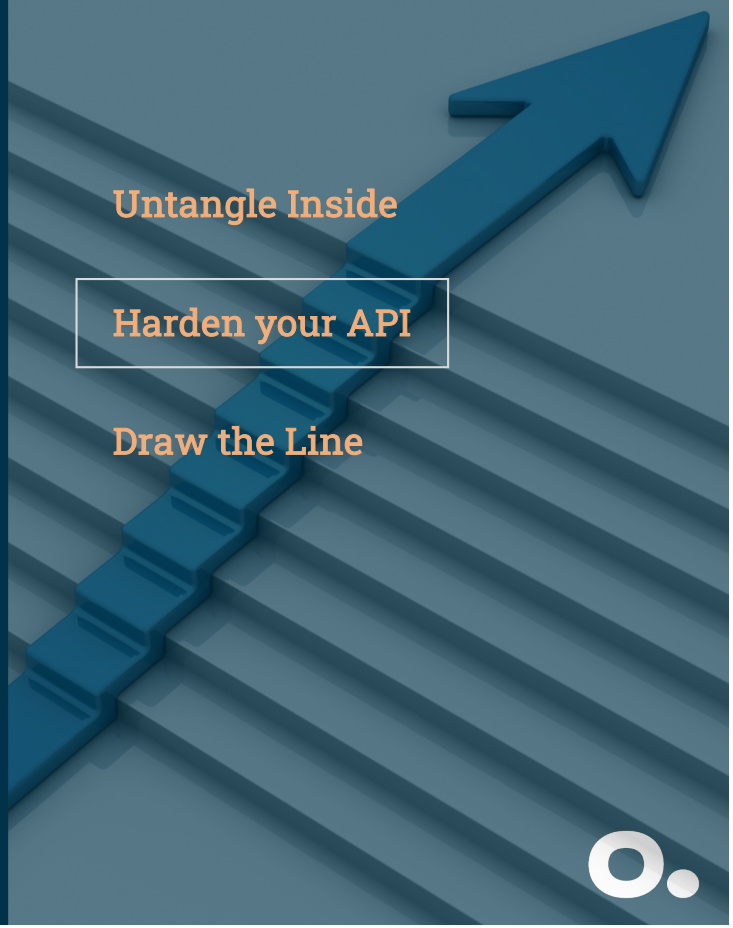
SECURE BY DESIGN

Expose Semantics – Enforce Data

```
public void checkout(final int roomNumber) {
   if (!RoomNumber.isValid(roomNumber)) {
      reporter.logInvalidRoomNumber(roomNumber);
   }

   houseKeepingService.registerForCleaning(roomNumber);
   minibarService.replenish(roomNumber);
   // other operations ...
}
```

```
public void checkout(final RoomNumber checkedOut) {
   var roomNumber = checkedOut.room();
   houseKeepingService.registerForCleaning(roomNumber);
   minibarService.replenish(roomNumber);
   // other operations ...
}
```

Evolution Steps



SECURE BY DESIGN

Trickle Down Semantics – Clarifying Data

```
public void checkout(final RoomNumber checkedOut) {
    var roomNumber = checkedOut.room();
    houseKeepingService.registerForCleaning(roomNumber);
    minibarService.replenish(roomNumber);
    // other operations ...
}
```

```
public void checkout(final RoomNumber checkedOut) {

    houseKeepingService.registerForCleaning(checkedOut);
    minibarService.replenish(checkedOut);
    // other operations ...
}
```

Evolution Steps

Untangle Inside

Harden your API

Draw the Line

Summary – Legacy

The Three Steps Evolution Model allows you to

- start small and stop at any point
- easily interleave with other work
- apply domain primitives in a controlled way

SECURE BY DESIGN

Confidentiality

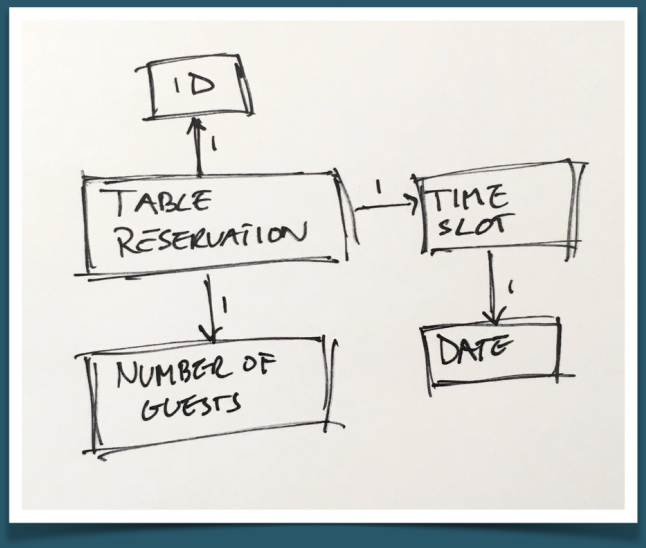
Lisa M Photography https://flic.kr/p/4saSBE

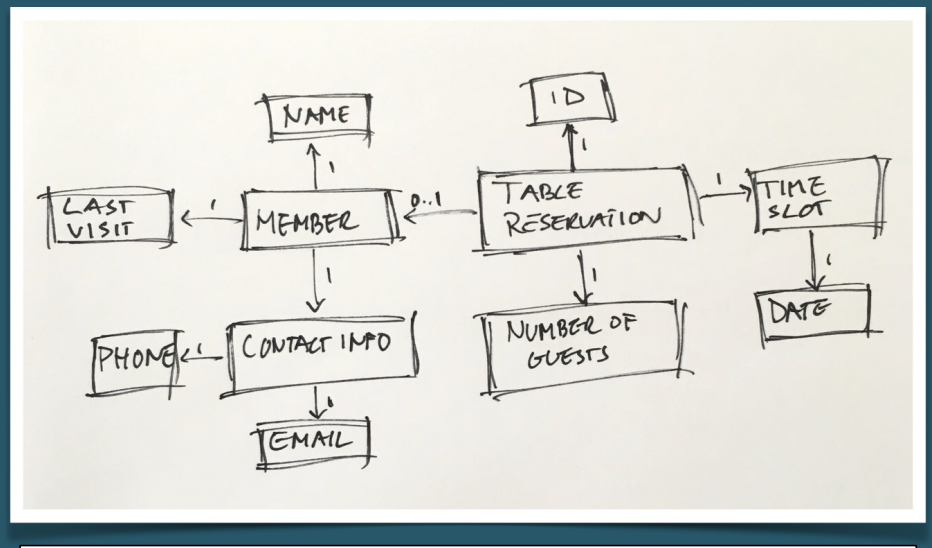
Example – Fetching a Table Reservation

```
public TableReservation fetch(final String reservationId) {

    logger.info("Fetching table reservation: " + reservationId);

    final TableReservation tableReservation = repository.reservation(reservationId);

    logger.info("Received " + tableReservation);

    return tableReservation;

}
```

- A table reservation is fetched by ID
- The reservation ID and the table reservation object are logged – is this intentionally or just debugging?
- Who cares – right?

Evolving Business Domain Model



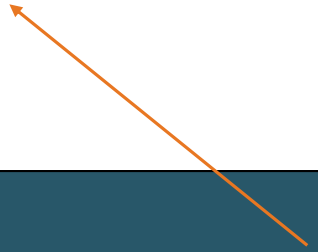


A table reservation is extended by info from the loyalty program to give better service – what could possibly go wrong?

SECURE BY DESIGN

Implicit Data Leakage

```
public TableReservation fetch(final String reservationId) {
    logger.info("Fetching table reservation: " + reservationId);
    final TableReservation tableReservation = repository.reservation(reservationId);
    logger.info("Received " + tableReservation);
    return tableReservation;
}
```



Implicit serialization may cause sensitive data leakage

SECURE BY DESIGN

Read-Once Pattern

- Sensitive data should be modeled explicitly
- Where and how sensitive data is consumed should be defined explicitly
- Using Read-Once (or read N-times) allows you to detect unintended data leakage!

```
...
    logger.info("Received " + tableReservation);

    return tableReservation;

}
```

```
public final class SensitiveValue implements Externalizable {
  private transient final AtomicReference<String> value;

  public SensitiveValue(final String value) {
    this.value = new AtomicReference<>(validate(value));
  }

  public String value() {
    return notNull(value.getAndSet(null), "Sensitive value has already been consumed");
  }

  private static String validate(final String value) {
    // Check domain-specific invariants
    return notBlank(value).trim();
  }

  @Override
  public String toString() {
    return "SensitiveValue{value=*****}";
  }

  @Override
  public void writeExternal(final ObjectOutput out) { deny(); }
  @Override
  public void readExternal(final ObjectInput in) { deny(); }

  private static void deny() {
      throw new UnsupportedOperationException("Not allowed");
  }
```

Summary – Confidentiality

Read-Once pattern supports confidentiality by

- aggressively detects unintended data leakage
- clarifies where sensitive data is consumed
- allows most errors to be caught before reaching production

SECURE BY DESIGN

Traceability

Tsuda https://flic.kr/p/8Kvzm

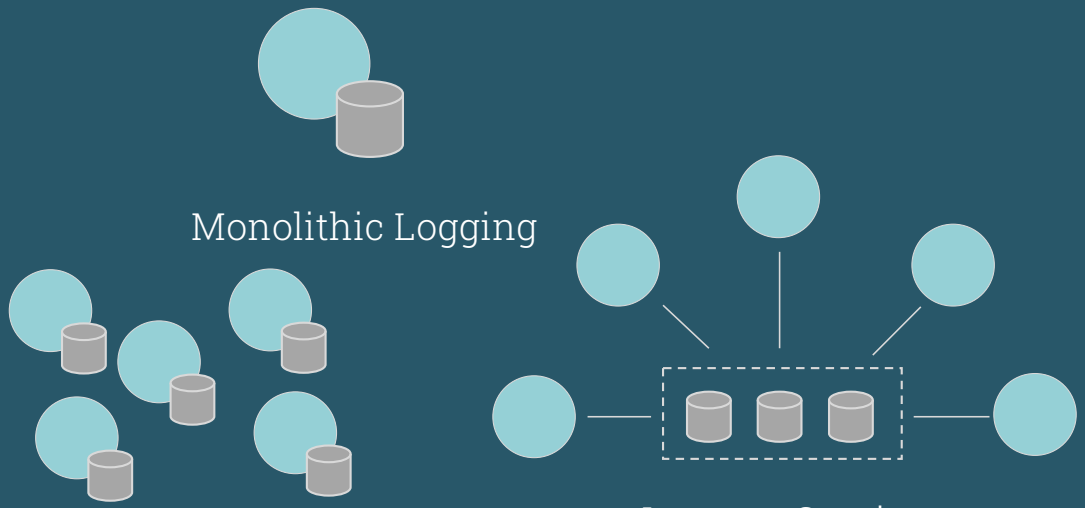
SECURE BY DESIGN

Traceability – Interesting Problems

- Accessibility – how is data accessed?
- Persistence – life cycle management
- Correlation – what happened when?
- Tampering – integrity of data
- Exploitability – 2nd order injection attacks

SECURE BY DESIGN

Evolution of Logging Strategies



Using a centralized log as a service yields

- Protection against log tampering
- Access Management (think GDPR)
- Post-termination log persistence
- Simplified log correlation

SECURE BY DESIGN

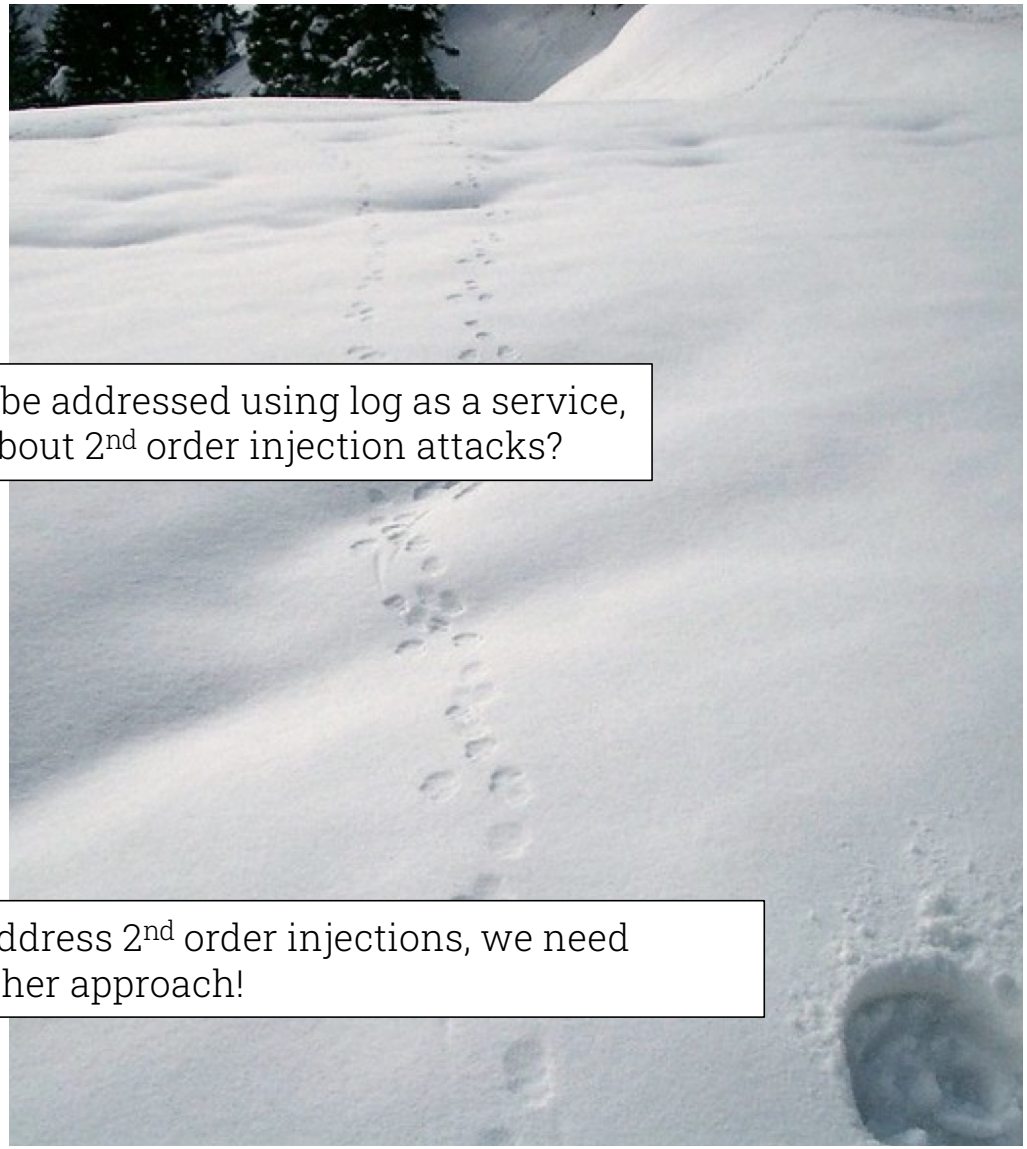
Traceability – Interesting Problems

- Accessibility – how is data accessed?
- Persistence – life cycle management
- Correlation – what happened when?
- Tampering – integrity of data

These can be addressed using log as a service, but what about 2nd order injection attacks?

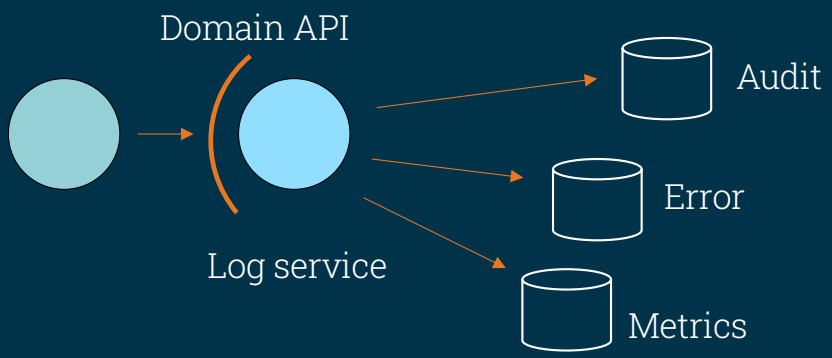
- Exploitability – 2nd order injection attacks

To address 2nd order injections, we need another approach!

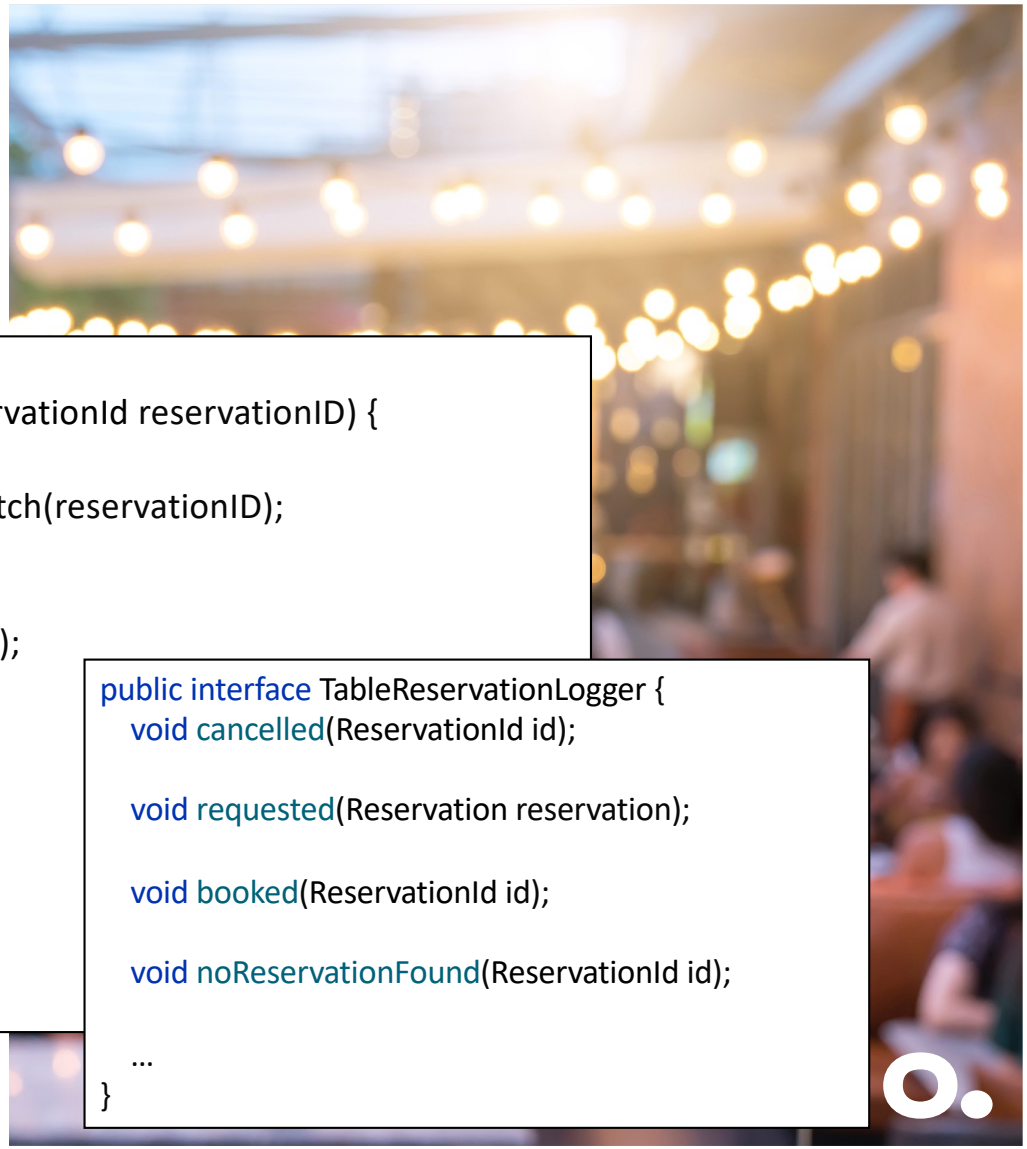


Domain Oriented Log API

- The log service implements a domain specific API that only accepts well defined objects
- Behind the scenes, the log service
  - Separates data based on purpose, e.g. Audit, Error, Metrics, etc
  - Needs to adapt to the peculiarities of the log system, e.g. apply proper encoding!
- This way, what we log becomes an active design decision!



Domain Oriented Log API – Logging a Table Reservation



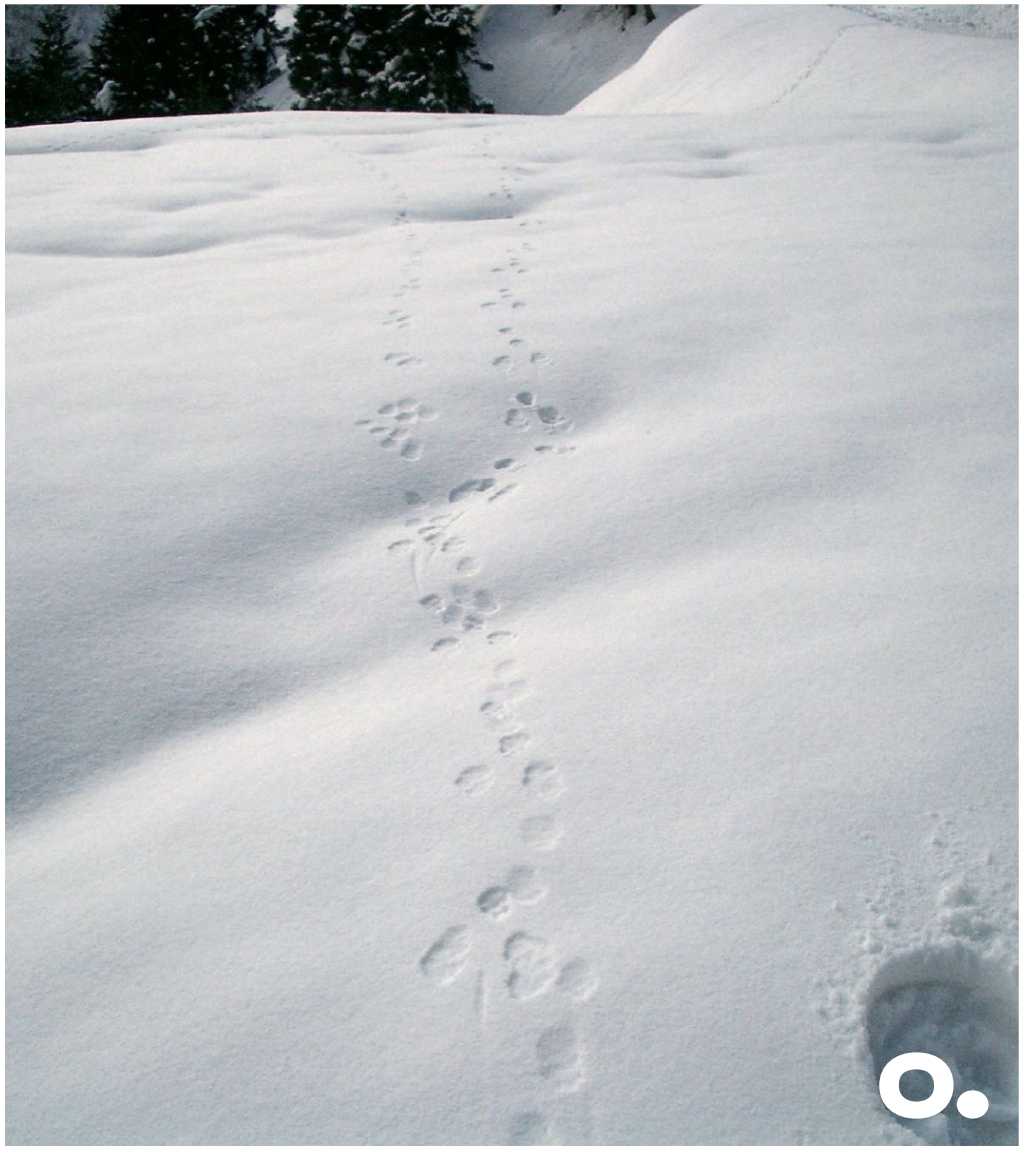
```
public Optional<Reservation> fetchReservation(final ReservationId reservationID) {

   final Reservation reservation = reservationRepository.fetch(reservationID);

   if (reservation == null) {
      reservationLogger.noReservationFound(reservationID);
      return Optional.empty();
   }

   reservationLogger.requested(reservation);

   return Optional.ofNullable(reservation);
}
```

```
public interface TableReservationLogger {
   void cancelled(ReservationId id);

   void requested(Reservation reservation);

   void booked(ReservationId id);

   void noReservationFound(ReservationId id);

   …
}
```

Summary – Traceability

Centralized logging and Domain-Oriented Log API support traceability by

- making logging into an explicit subsystem
- turning logging into an active design decision
- clarifies logging for different purposes



SECURE BY DESIGN

Availability

Availability %	Downtime per year[note 1]	Downtime per month	Downtime per week	Downtime per day
55.5555555% ("nine fives")	162.33 days	13.53 days	74.92 hours	10.67 hours
90% ("one nine")	**36.53 days**	**73.05 hours**	**16.80 hours**	**2.40 hours**
95% ("one nine five")	18.26 days	36.53 hours	8.40 hours	1.20 hours
97%	10.96 days	21.92 hours	5.04 hours	43.20 minutes
98%	7.31 days	14.61 hours	3.36 hours	28.80 minutes
99% ("two nines")	**3.65 days**	**7.31 hours**	**1.68 hours**	**14.40 minutes**
99.5% ("two nines five")	1.83 days	3.65 hours	50.40 minutes	7.20 minutes
99.8%	17.53 hours	87.66 minutes	20.16 minutes	2.88 minutes
99.9% ("three nines")	**8.77 hours**	**43.83 minutes**	**10.08 minutes**	**1.44 minutes**
99.95% ("three nines five")	4.38 hours	21.92 minutes	5.04 minutes	43.20 seconds
99.99% ("four nines")	**52.60 minutes**	**4.38 minutes**	**1.01 minutes**	**8.64 seconds**
99.995% ("four nines five")	26.30 minutes	2.19 minutes	30.24 seconds	4.32 seconds
99.999% ("five nines")	**5.26 minutes**	**26.30 seconds**	**6.05 seconds**	**864.00 milliseconds**
99.9999% ("six nines")	**31.56 seconds**	**2.63 seconds**	**604.80 milliseconds**	**86.40 milliseconds**
99.99999% ("seven nines")	**3.16 seconds**	**262.98 milliseconds**	**60.48 milliseconds**	**8.64 milliseconds**
99.999999% ("eight nines")	**315.58 milliseconds**	**26.30 milliseconds**	**6.05 milliseconds**	**864.00 microseconds**
99.9999999% ("nine nines")	**31.56 milliseconds**	**2.63 milliseconds**	**604.80 microseconds**	**86.40 microseconds**

Advanced Persistent Threats

Advanced Persistent Threats (APT)

Sophisticated, long-term cyberattacks where an intruder gains — and maintains — unauthorized access to a network over an extended period of time.

Advanced – attackers use sophisticated techniques.

Persistent – the goal isn't a smash-and-grab; attackers stay in the environment for months or years

Threat – significant data loss or damage

Who typically drives APTs?

- Predominantly nation-state actors and well-funded criminal organizations, e.g. APT28 (Russia), APT41 (China), and Lazarus Group (North Korea?).

Image: Richard Patterson, https://flic.kr/p/24GcRZQ

omega point.

Claude Mythos

Announced April 7th, 2026

- Anthropic's most capable model to date — not publicly released (yet)
- Its ability to autonomously find high-severity vulnerabilities in major operating systems and browsers makes it a potential weapon for cybercriminals and state actors.

AISI Expert CTF

Offensive security tasks drawn from expert-level hacking challenges, carefully graded by difficulty — where the model must identify and exploit vulnerabilities in target systems to retrieve a hidden flag.

73%

Previous models score: **0%**

SWE-bench Verified

Real-world engineering tasks drawn from actual GitHub issues, carefully human-verified to ensure each problem is solvable and the success criteria are unambiguous.

93.9%

SWE-bench Pro (public)

Real-world engineering tasks sourced from GPL-licensed repositories, carefully designed to resist contamination by using standardized scaffolding so no model can game results through custom agent tuning.

77.8%

SWE-bench Pro (private)

Proprietary codebases from startup partners that are not publicly accessible, making it the most realistic test of how a model performs on code it has genuinely never seen before.

?

omega point.

Stale Environments – utilized by APT

Culture

Myths & Misconceptions

Organizational Silos

Budgets

Application Design

Ceremony

Fear

Image: George Fryd, https://flic.kr/p/jV1L6C

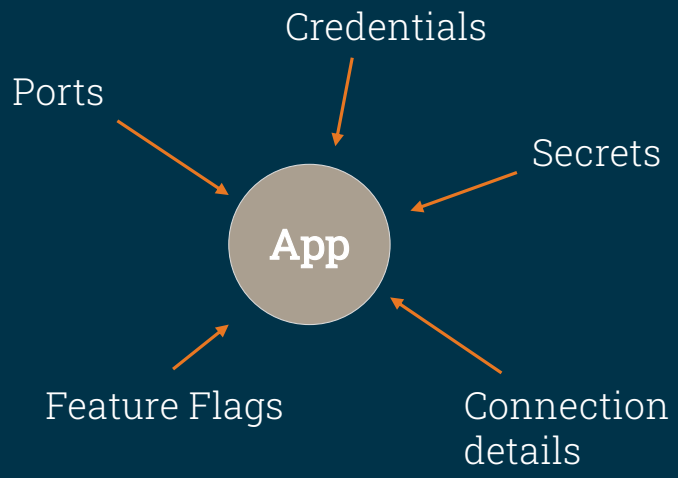
omega point.

Inspiration from the Cloud

- Externalized Configuration
- Immutable Builds
- Service Discovery
- Stateless Services
- Expand/Contract API

Externalized Configuration

- Configuration that changes between environments belongs to the environment
- Makes your application environment agnostic
- Challenge: could you open-source your codebase right now without exposing any secrets?



omega point.

Immutable Builds

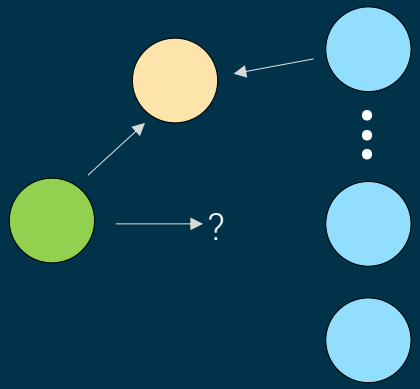
- Build once run anywhere
- Never modify an artifact – always build new
- **Authoritative Build**
  - What's tested is what you run
  - Fully defined by repository & artifactory content
  - Sign your images!
- **Automatic Pipelines**
  - Minimizes risk of human error
  - Repeatability

Spencer Cooper, https://flic.kr/p/cp5WgS

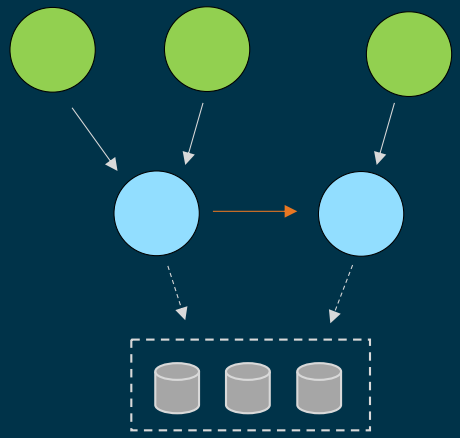
Ryan McFarland, https://flic.kr/p/4scuyw

omega point.

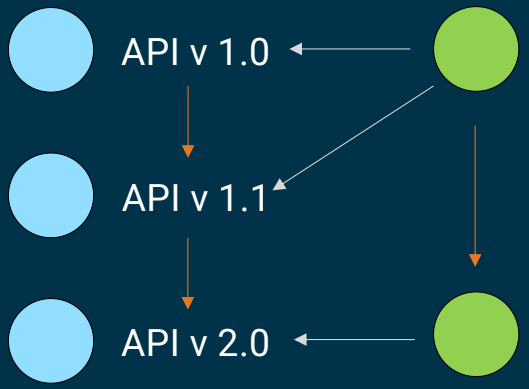
Designing for Flexibility



Service Discovery



Stateless Services



Expand / Contract APIs

omega point.

3 Rs of Enterprise Security

Rotate

Repave

Repair

Attribution: Justin Smith, Pivotal

omega point.

Rotate

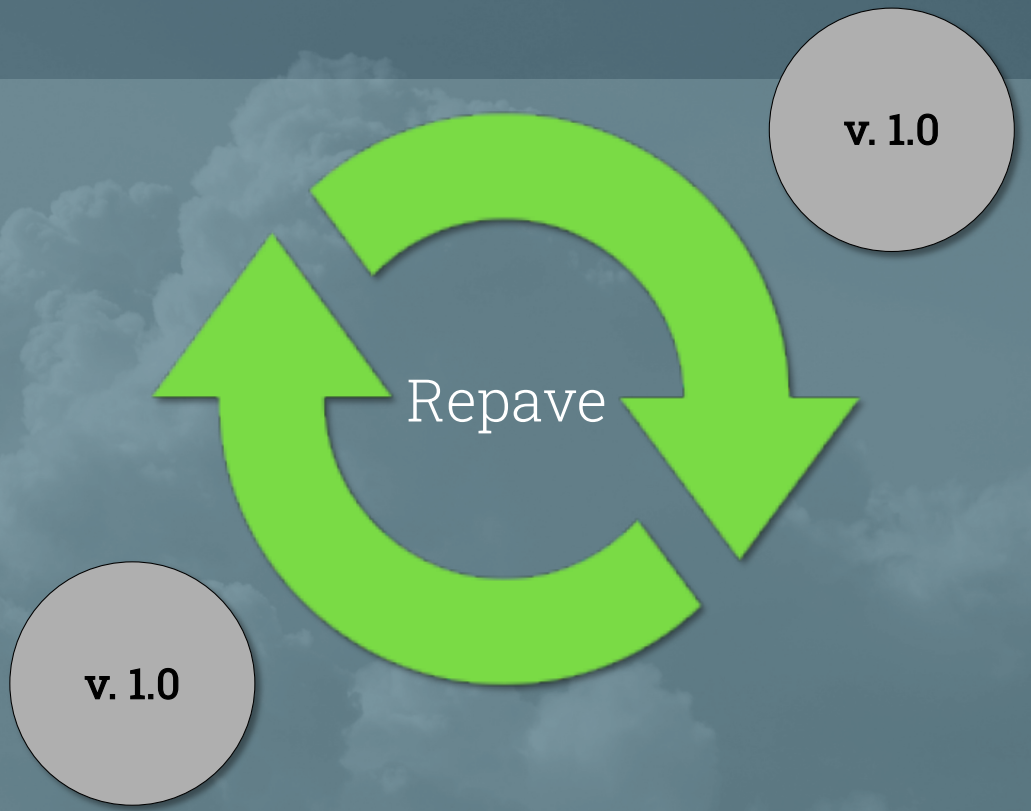
- By externalizing configuration, secrets can be placed in a vault that allows automatic rotation of secrets.
- But how does “Rotate” address APT?

Repave

- Combining
  - immutable builds
  - service discovery
  - seamless deployments

makes it possible to repave applications in an automatic fashion without downtime.

- But how does “Repave” address APT?



Repair

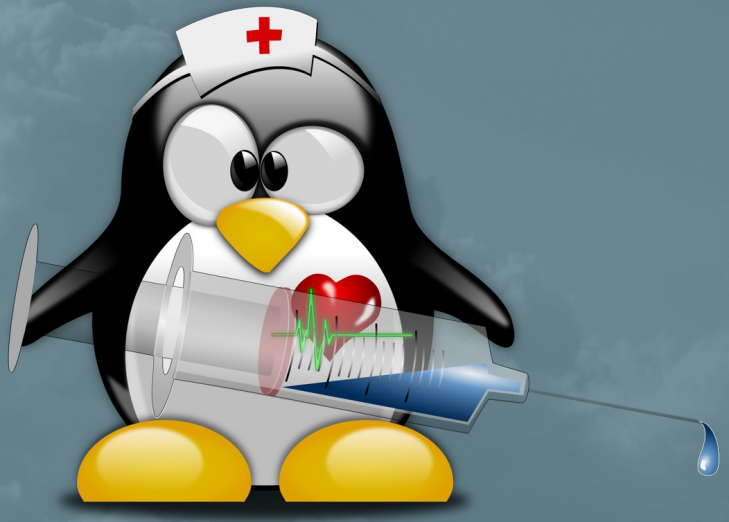
By combining

- build pipelines (with automated tests)
- immutable builds
- service discovery
- seamless deployments

it becomes possible to continuously update the OS image and verify that it works with the application(s).

An updated OS image can then be rolled out during the next repavement phase or on a scheduled basis.

But how does ”Repair” address APT?



https://freesvg.org/tux-nurse-vector-illustration

omega point.

Summary – Availability

The architectural patterns support availability by

- promoting zero downtime
- enabling automatic patch management
- lowering the threshold for rotating secrets

Summary

- A lot of security vulnerabilities are due to bugs
- Bugs can be prevented by software design
- Secure by Design collects designs that prevents bugs that manifest as security vulnerabilities


Q & A