Rethinking Threat Modelling for Dev Teams: a Scalable Approach

A lightweight, developer-centric approach to integrate threat modelling into the development workflow, reducing reliance on security experts.

Andrea Scaduto Co-founder & Director, SecureFlag

Andrew Hainault Managing Director, LevelBlue

SecureFlag | LevelBlue

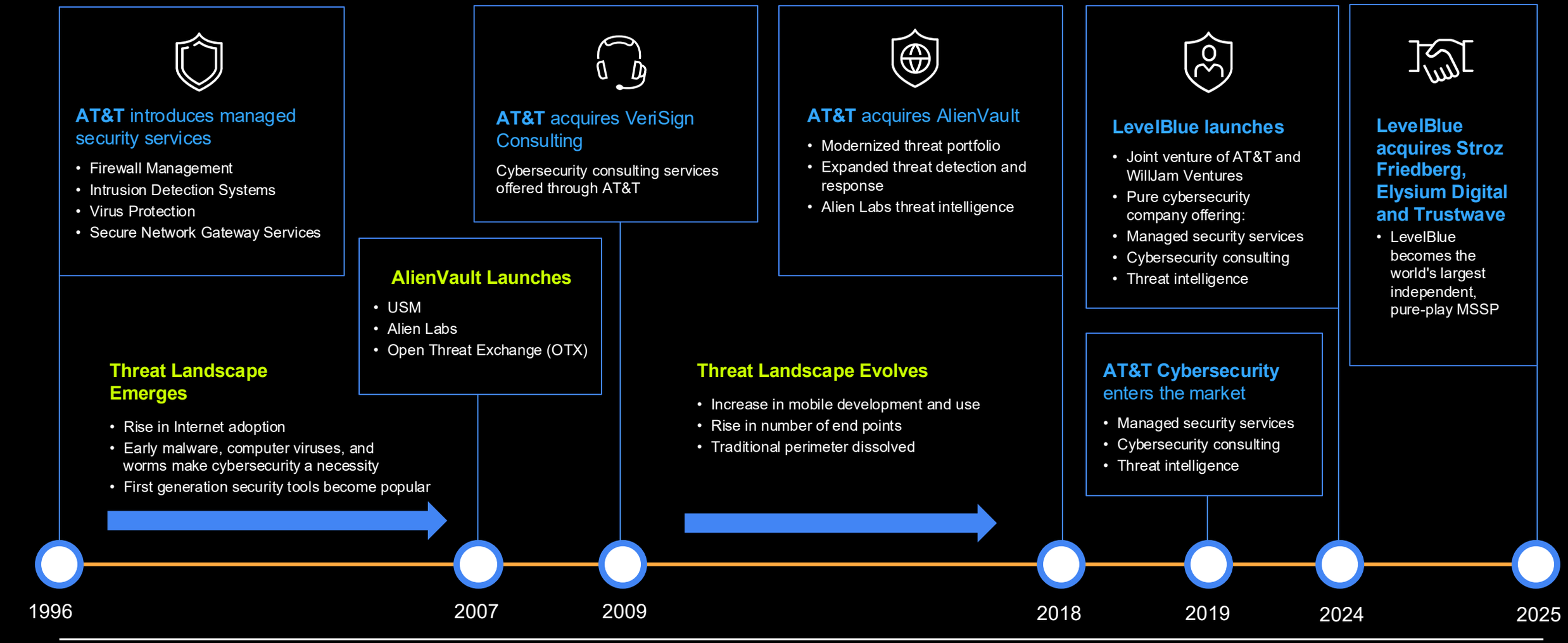
Andrew Hainault

- Managing Director, Cybersecurity Advisory @ **LevelBlue**
- Secure Design & Architecture, Threat Modelling

Andrea Scaduto

- Co-founder & Director @ **SecureFlag**
- Secure Coding, Threat Modelling, Penetration Testing

LevelBlue – A History of Excellence

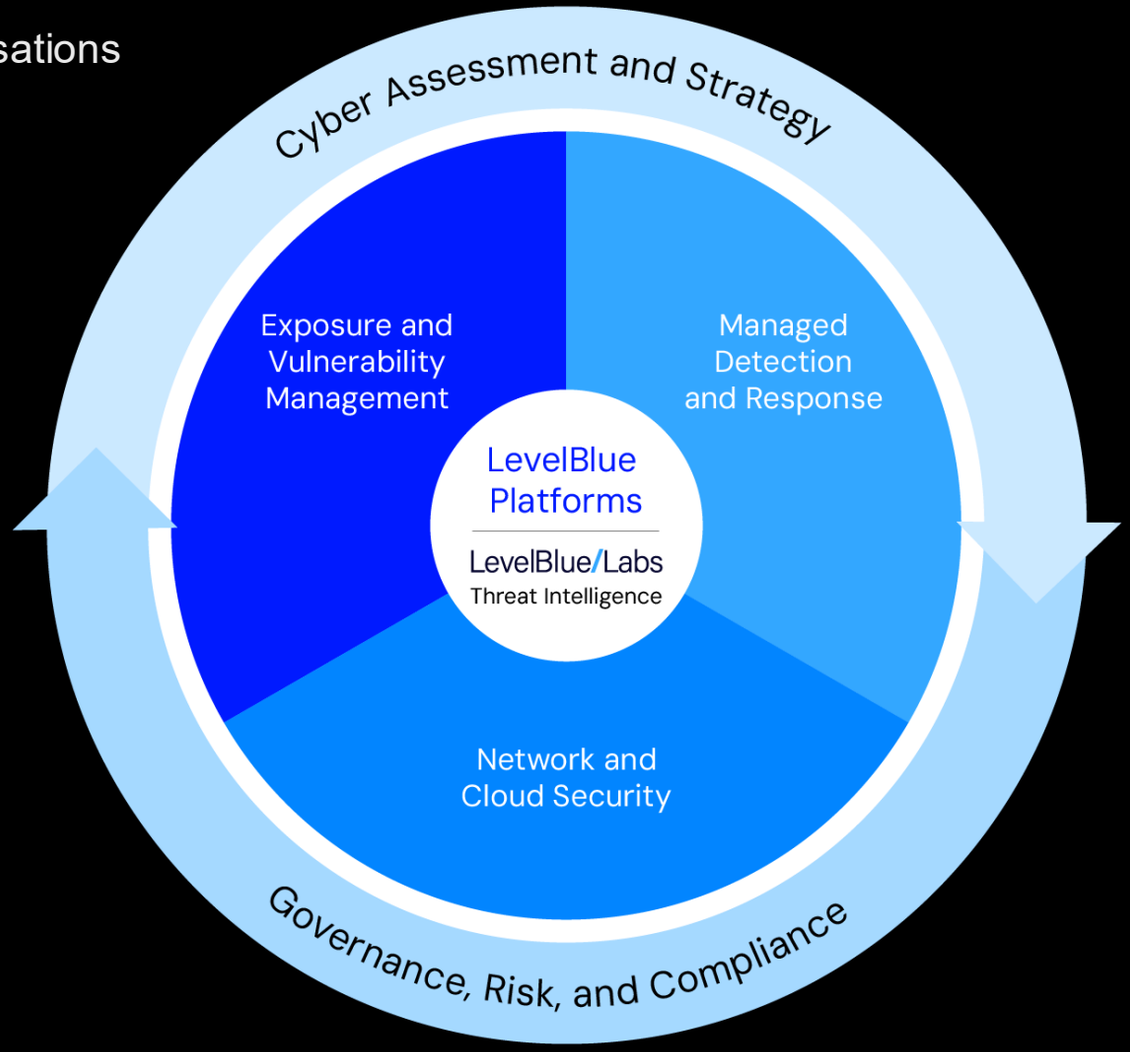


LevelBlue

LevelBlue

We Simplify Cybersecurity for Businesses and Organisations

- **Consulting Services** to benchmark, design, and transform your operations and compliance programs
- **Network Security** to consolidate and integrate your network and security solutions
- **Exposure Management** to understand your risk levels and operationalize vulnerability management
- **Detection and Response** to detect, prioritize, and respond to malicious activities



Threat Modelling:
What, When, and Why It Matters

What Is Threat Modelling?

- A systematic approach to identify, analyse, and mitigate potential security threats.
- Involves understanding a system's assets, potential attackers, and attack vectors.
- Helps teams anticipate vulnerabilities before they become real risks.

When Should It Be Done?

- **Early in Development:** Ideally, during the design phase, to ensure security is built in from the start.
- **Throughout the Lifecycle**: Revisit whenever there are significant changes.

Why Is It Important?

- **Proactive Security**: Identifies potential issues before they can be exploited, reducing the cost and impact of breaches.
- **Informed Decisions**: Prioritises risks so teams can allocate resources effectively.
- **Regulatory Compliance**: Demonstrates due diligence and can help meet legal or industry security requirements.

Traditional **Threat Modelling**

Define Scope

- Establish objectives and scope for the threat modelling exercise.

Diagram the Architecture

- Create a visual representation to understand how components interact.

Identify Threats

- Use structured techniques or frameworks (e.g., STRIDE, PASTA) to enumerate potential security threats.

Evaluate and Prioritise Risks

- Assess the impact of each threat and assign risk ratings.

Develop Mitigation Strategies

- Propose countermeasures or design changes to reduce or eliminate identified threats.

Document and Review

- Keep a record of identified threats, associated risks, and mitigations.
- Regularly revisit and update the threat model as the system evolves or new threats emerge.

Challenges with Traditional Threat Modelling

Scope often includes the entire application/system.

- It requires involving people who know the architecture of the entire system.
- The scope is very broad, which fails to provide actionable results.
- Typically, it involves application leads, while most developers are never exposed to threat modelling.

Threat enumeration requires specialised security knowledge.

- organisations often lack the specialised security expertise required for traditional threat modelling methods.
- Reliance on security experts creates bottlenecks, limiting scalability in large organisations.

Activity is resource intensive, producing limited value.

- Threat modelling is often performed after the application is built.
- Typically produces hundreds of threats.

Developer-Driven Methodology

Scope

- Can be applied to one or more components or individual pieces of functionality.

Audience

- Designed for use by individual developers or small teams.

Threat Templates

- Uses a concise set of lists of up to 20 prioritised threats and controls for different contexts.

Threat Templates: the core enabler

Predefined collections of (max 20) threats and controls tailored to specific contexts/stakeholders:

- Implementation (e.g., OWASP Top 10, language-specific secure coding guidelines).
- Deployment Environment (e.g., AWS, Azure, GCP, on-premises, embedded devices).
- Compliance Standards (e.g., FedRAMP, PCI DSS, HIPAA, GDPR).
- Organisational Priorities (e.g., patterns of vulnerabilities).

Threat Templates act as a checklist for developers to identify the most important threats:

- Provides description, default severity rating and applicable controls for each threat.
- Multiple Threat Templates can be used by different stakeholders for the same threat model.
- Security teams should tailor Threat Templates for the organisation's specific priorities.

Secure Code Implementation Threat Template (example)

Threat	Description	Severity	Suggested Controls
Insecure Direct Object References (IDOR)	Exposing references to internal objects references allowing attackers to access unauthorised data.	High	- Implement access control checks at every layer. - Use randomised or hashed references. - Validate user permissions before returning data.
Server-Side Request Forgery (SSRF)	Trick the server into making unintended requests to internal services exposing sensitive resources/information.	Moderate	- Validate and sanitise all URLs or endpoints. - Restrict outbound traffic with allowlists/firewalls. - Disable unneeded protocols.
...	...	...	...

AWS Threat Template (example)

Threat	Description	Severity	Suggested Controls
S3 Bucket Misconfiguration	S3 buckets left publicly readable or writable, overly permissive Access Control Lists (ACLs) or bucket policies.	High	- Use S3 Block Public Access settings. - Enforce least-privilege IAM policies for bucket access. - Regularly audit S3 permissions with AWS Config or third-party tools.
Insecure VPC Configuration	Poorly configured VPCs can expose internal services and data to the public internet.	Moderate	- Separate public/private subnets, use NAT Gateway or VPC Endpoints. - Restrict inbound/outbound traffic with ACLs and Security Groups. - Monitor and audit VPC Flow Logs for suspicious traffic.
...	...	...	...

PCI DSS Threat Template (example)

Threat	Description	Severity	Suggested Controls
Unauthorised Access to Cardholder Data	Unauthorised access to stored cardholder data through inadequate access controls.	High	- PCI DSS Rq. 7: Implement strong access control measures.. - PCI DSS Rq. 8: Enforce unique user IDs and strong authentication. - PCI DSS Rq. 10: Maintain comprehensive audit trails of all access.
Data Retention Beyond Requirements	Cardholder data is retained longer than necessary, increasing the risk of exposure if compromised.	Moderate	- PCI DSS Rq. 3.1: Limit data retention to what is necessary for business, legal, and regulatory requirements.
...	...	...	...

Building Effective **Threat Templates**

- Developed by security experts to ensure threats and controls align with compliance standards, application environments, and organisational needs.
- Focus on the most important risks (e.g. 20) so the entire list can be quickly read and understood.
- Enables developers to skip extensive brainstorming and focus on a curated, relevant threat list.
- Derive threats and controls from standards like PCI DSS, CIS Benchmarks, NIST CSF, and map controls to NIST SP 800-53.

Rapid **Developer-Driven** Threat Modelling Methodology (RaD-TM)

Phases of the Methodology

1. Graphical Representation: map components, interactions, and data flows.
2. Identify Trust Boundaries: highlight transitions between systems or privilege levels.
3. Identify Threats: use **Threat Templates** to uncover relevant risks.
4. Map Controls: select and evaluate controls from the Threat Template.
5. Assess Status: mark threats as mitigated, unmitigated, or accepted.

SecureFlag® | LevelBlue

Example

...

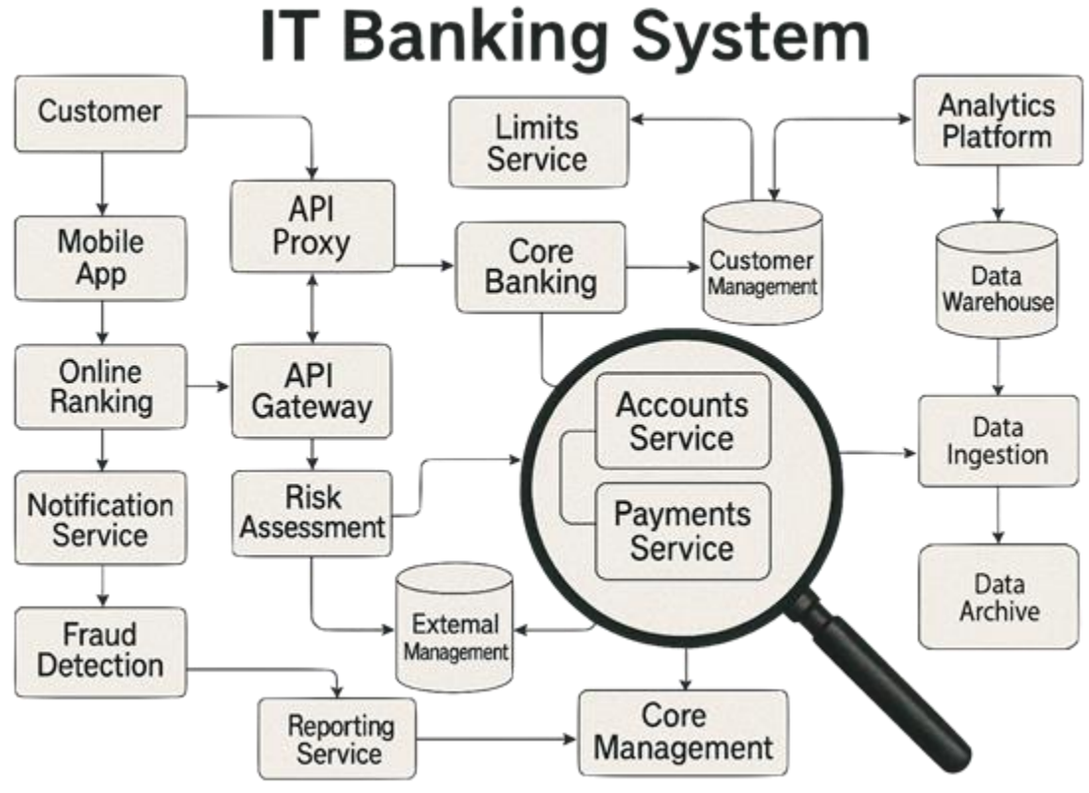
Define **Scope**

Traditional approach often scopes the *entire system*

- Involves many stakeholders → time-consuming and hard to update
- Broad scope dilutes focus → results often vague and not actionable

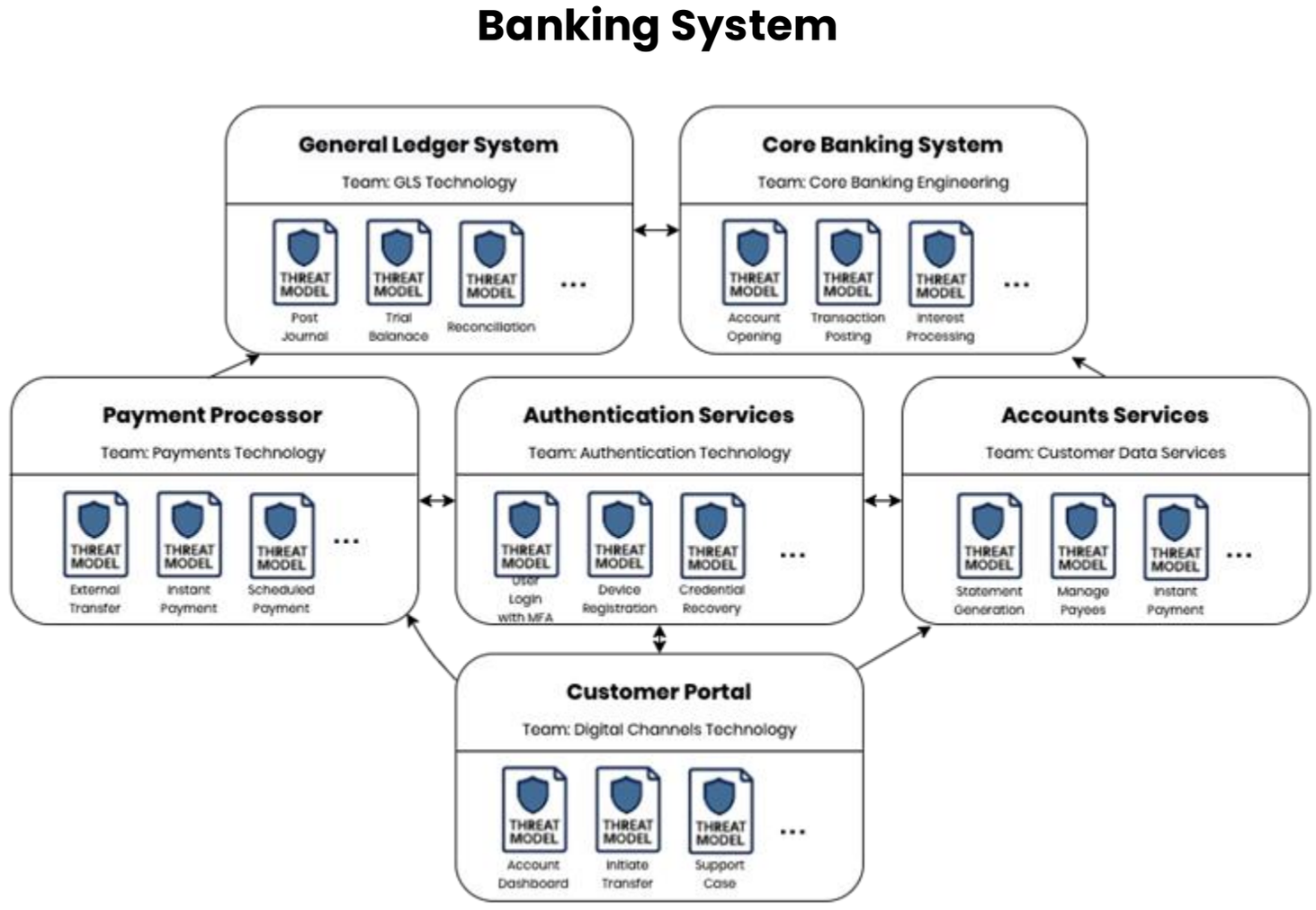
Shift towards **component** or **feature-based threat models**

- Narrower scope → deeper, more detailed analysis
- Easier to repeat and update as features evolve
- Ownership by development teams → better integration into workflows and accountability



Decompose **and Delegate**

- The system is composed of multiple interconnected components.
- Each component has *key* flows and functionalities that are analyzed separately.
- Development teams create threat models for the flows they own.
- Linking these models across components provides a realistic model of the entire system.



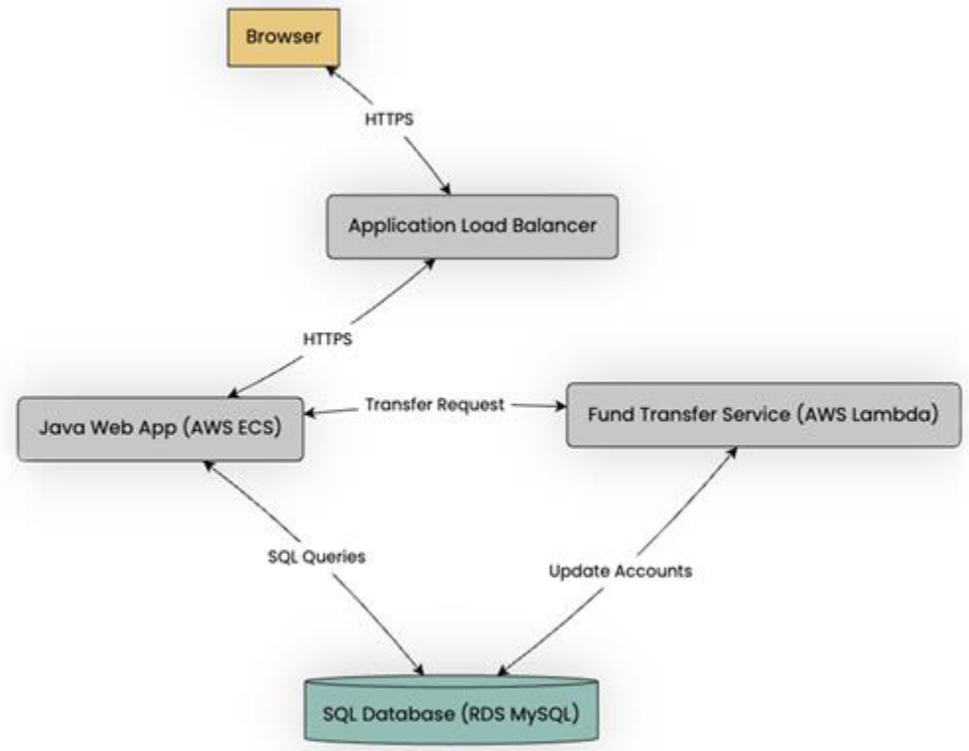
1. Produce a **Graphical Representation**

Feature

- Allows users to transfer funds between their accounts.

Inputs and Outputs

- Inputs
  - Logged-in User
  - Source and destination accounts
  - Transfer amount
- Outputs
  - Transaction confirmation
  - Error messages



Feature Use Case (High-Level)

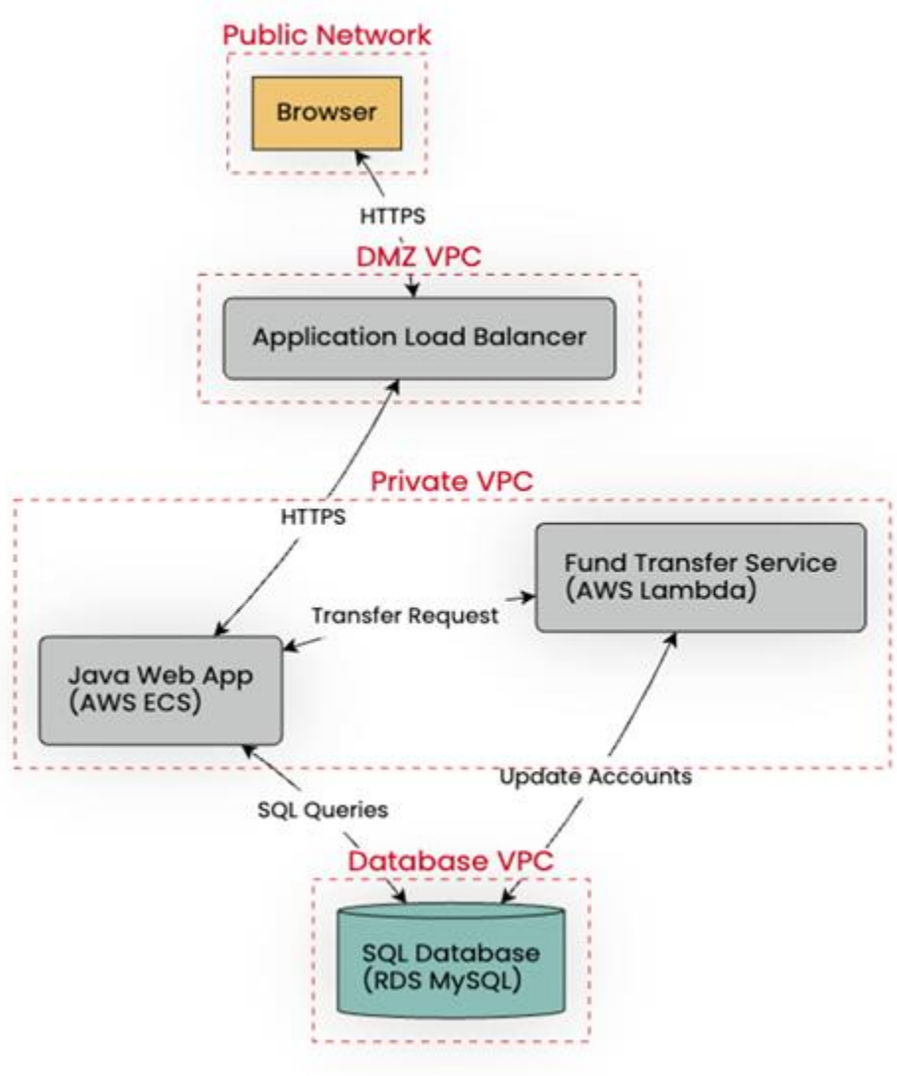
1. User logs in with valid credentials.
2. User navigates to “Transfer Funds.”
3. User selects source account, destination account, and amount.
4. Application validates the input
5. Application makes transfer, updates database, and returns a confirmation message.

2. Identify **Trust Boundaries**

- Define where different security levels meet within a system
- Separate trusted and untrusted systems, networks, or components
- Keep it simple…

Identified Trust Boundaries:

- Public Network → DMZ VPC
- DMZ VPC → Private VPC
- Private VPC → Database VPC



3. Identify **Threats**

Audience: **Developer** Threat Template: **Secure Code Implementation**

Threat	**Description**	**Default Severity**	**Affected Components**
Insecure Direct Object References (IDOR)	References to internal objects could allow attackers to access unauthorised data.	High	- Java Web App
Server-Side Request Forgery (SSRF)	Trick the server into making unintended requests to internal services exposing sensitive resources/information.	Moderate	- Java Web App - Application Load Balancer
.....	.....	.....	.....

Audience: **Cloud Engineer** Threat Template: **AWS**

Threat	**Description**	**Default Severity**	**Affected Components**
S3 Bucket Misconfiguration	S3 buckets left publicly readable or writable, overly permissive Access Control Lists (ACLs) or bucket policies.	High	N/A
Insecure VPC Configuration	Poorly configured VPCs can expose internal services and data to the public internet.	Moderate	- Java Web App - Fund Transfer Service - SQL Database
...	.....	...	...

Audience: **Risk Owner** Threat Template: **PCI DSS**

Threat	**Description**	**Default Severity**	**Affected Components**
Unauthorised Access to Cardholder Data	Unauthorised access to stored cardholder data through inadequate access controls.	High	- SQL Database
Data Retention Beyond Requirements	Cardholder data is retained longer than necessary, increasing the risk of exposure if compromised.	Moderate	- SQL Database
...	...	...	...

4. Map Controls

Audience: **Developer** Threat Template: **Secure Code Implementation**

Threat	...	Affected Components	Suggested Controls	Controls Implemented?	Rationale
Insecure Direct Object References (IDOR)	...	- Java Web App	- Implement access control checks at every layer. - Use randomised or hashed references. - Validate user permissions before returning data.	Yes	Authorisation checks are comprehensively implemented,, randomised references used.
Server-Side Request Forgery (SSRF)	...	- Java Web App - Application Load Balancer	- Validate and sanitise all URLs or endpoints. - Restrict outbound traffic with allowlists/firewalls. - Disable unneeded protocols.	Partially	The load balancer is correctly configured, but the Java web application does not validate input when constructing the internal URL
-	-	-	-	-	-

Audience: **Cloud Engineer** Threat Template: **AWS**

Threat	...	Affected Components	Suggested Controls	Controls Implemented?	Rationale
Insecure VPC Configuration	...	- Java Web App - Fund Transfer Service - SQL Database	- Separate public/private subnets, NAT Gateway or VPC Endpoints. - Restrict inbound/outbound traffic with ACLs and Security Groups. - Monitor and audit VPC Flow Logs for suspicious traffic.	Yes	Three separate VPCs are in place, and all security groups are correctly configured.
-	-	-	-	-	-

Audience: **Risk Owner** Threat Template: **PCI DSS**

Threat	...	Affected Components	Suggested Controls	Controls Implemented?	Rationale
Unauthorised Access to Cardholder Data	...	- SQL Database	- PCI DSS Rq. 7: Implement strong access control measures.. - PCI DSS Rq. 8: Enforce unique user IDs and strong authentication. - PCI DSS Rq. 10: Maintain comprehensive audit trails of all access.	Yes	Robust authentication is implemented, and all access is logged and monitored.
Data Retention Beyond Requirements	...	- SQL Database	- PCI DSS Rq. 3.1: Limit data retention to what is necessary for business, legal, and regulatory requirements.	Partially	No data retention policy in place, application data is stored indefinitely. Cardholder data removed after use.
...	...	...	...	...	...

5. Assess **Status & Severity**

Audience: **Developer** Threat Template: **Secure Code Implementation**

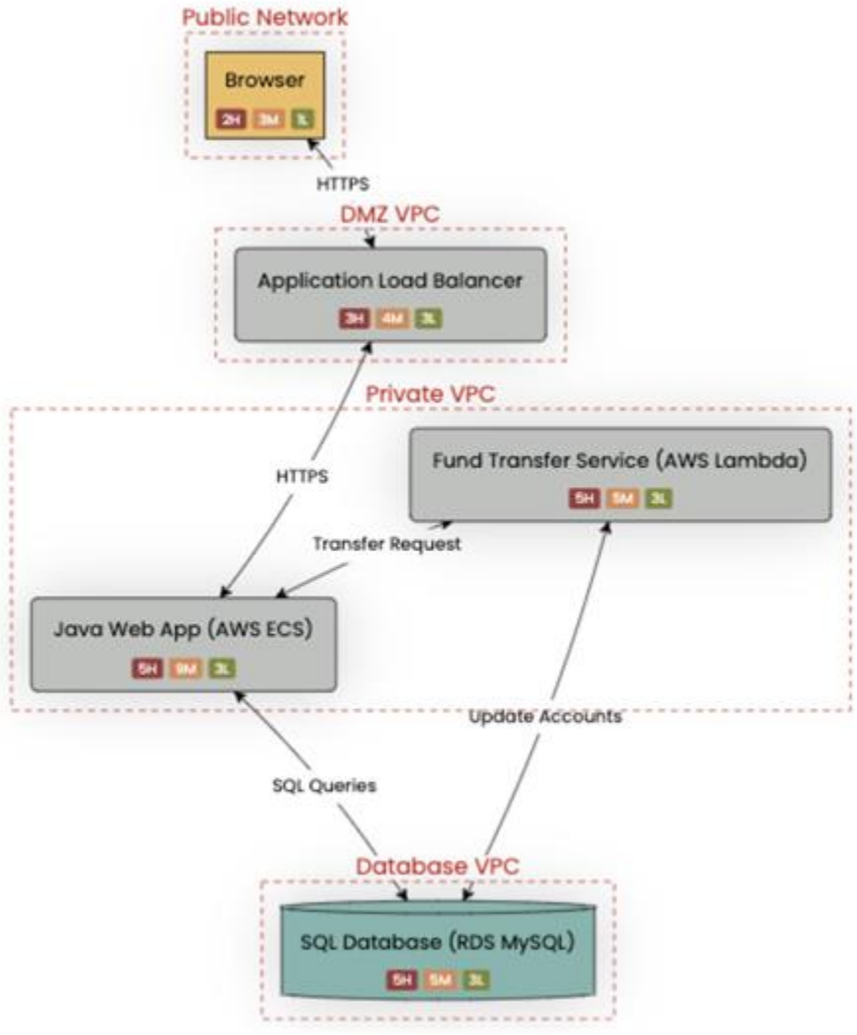
Threat	**...**	**Affected Components**	**Rationale & Next Steps**	**Status**	**Severity**
Insecure Direct Object References (IDOR)	...	- Java Web App	Authorisation checks are comprehensively implemented, and randomised references are employed.	Mitigated	-
Server-Side Request Forgery (SSRF)	...	- Java Web App - Application Load Balancer	The load balancer is correctly configured, but the Java web application does not validate input when constructing the internal URL. Input validation will be included in the next design sprint.	Open	Moderate
...	...	...	...	...	...

Audience: **Cloud Engineer** Threat Template: **AWS**

Threat	**...**	**Affected Components**	**Rationale & Next Steps**	**Status**	**Severity**
Insecure VPC Configuration	...	- Java Web App - Fund Transfer Service - SQL Database	Three separate VPCs are in place, and all security groups are correctly configured.	Mitigated	-
...	...	...	...	...	...

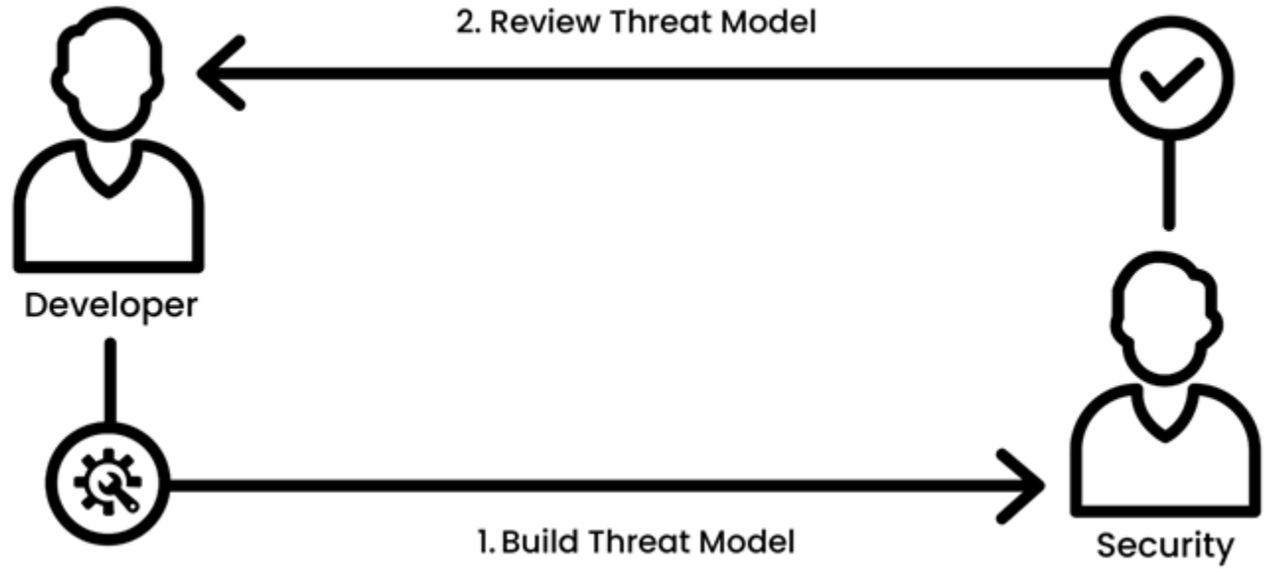
Audience: **Risk Owner** Threat Template: **PCI DSS**

Threat	**...**	**Affected Components**	**Rationale & Next Steps**	**Status**	**Severity**
Unauthorised Access to Cardholder Data	...	- SQL Database	Robust authentication is implemented, and all access is logged and monitored.	Mitigated	-
Data Retention Beyond Requirements	...	- SQL Database	No data retention policy is in place, and data is stored indefinitely. Only application data is stored indefinitely, while cardholder data and PII are retained only as necessary.	Accepted	Low
...	...	...	...	...	...



Threat	Component	Severity Rating
Abuse of Inadequate Authentication	Fund Transfer Service (AWS Lambda)	high
Abuse of Inadequate Authorization	Fund Transfer Service (AWS Lambda)	high
Code Injection	Fund Transfer Service (AWS Lambda)	high
Password Brute-Force	Fund Transfer Service (AWS Lambda)	high
Reuse of Known Credentials	Fund Transfer Service (AWS Lambda)	high
Abuse of Inadequate Authentication	Java Web App (AWS ECS)	high
Abuse of Inadequate Authorization	Java Web App (AWS ECS)	high
Code Injection	Java Web App (AWS ECS)	high
Password Brute-Force	Java Web App (AWS ECS)	high
Reuse of Known Credentials	Java Web App (AWS ECS)	high
Abuse of Inadequate Authentication	SQL Database (RDS MySQL)	high
Abuse of Inadequate Authorization	SQL Database (RDS MySQL)	high
Code Injection	SQL Database (RDS MySQL)	high
Password Brute-Force	SQL Database (RDS MySQL)	high
Reuse of Known Credentials	SQL Database (RDS MySQL)	high
Abuse of Misconfigured or Debug Feature	Application Load Balancer	mod
Denial of Service	Application Load Balancer	mod
Man-in-the-middle Attack	Application Load Balancer	mod
Vulnerable and Outdated Component	Application Load Balancer	mod

Scaling Threat Modelling



Automating Threat Modelling (... to a point)

Model Drawing

- Automates the creation of system diagrams and data flows
- Speeds up continuous refinement of the model alongside the evolution of the system.

Threat Enumeration

- Suggests relevant attack vectors and scenarios.
- Allows the incorporation of multiple Threat Templates for different stakeholders

Controls Mapping

- Aligns threats with appropriate security controls.
- Simplifies compliance and ensures consistent security coverage.

Scalable Threat Modelling

- Train development teams on lightweight threat modelling techniques.
- Shift from monolithic system-wide threat models to smaller, component-based models that dev teams own.
- Provide standardised threat templates so dev teams can identify key threats and apply approved controls without constant security oversight.
- Assign dev teams to maintain threat models, with security teams conducting asynchronous reviews.
- Integrate automated threat modelling tools in the SDLC to reduce manual overhead.

Adopt & Contribute

- Review the methodology & adopt it in your organisation (start with a *friendly* team)
- Contribute to enhance the RaD-TM methodology
- Contribute a Threat Template

github.com/secureflag/rad-tm

SCAN ME

SecureFlag® | LevelBlue

Q&A