CyberSecurity Basics: Lecture [H0Q24a]

Security Through the Software Lifecycle [H0Q31a]

Prof. Mathy Vanhoef

DistriNet – KU Leuven – Belgium

Info for CyberSecurity Basics [H0Q24a]

- Lecture on Friday at 9:30-10:50 on Cybersecurity Governance and Operations → cancelled due to illness
  - Recording of last year is on Toledo
- Keep Monday 9 October free for the project. More info will be given by Bart Preneel tomorrow.

Security Through the Software Lifecycle

This course studies security throughout the software lifecycle:

- From design → development → maintenance
- Intended for people with basic understanding of computer architecture and system software

We can’t cover everything

- Course gives a **high-level overview**...
- …and studies **selected aspects in detail**
- Goal: know what processes exists to improve security

Class Structure

There are ~12 lectures planned:

- High-level content but also deep-dives into selected areas

There will be one assignment:

- Case study on pentesting, threat modelling, **fuzzing**,…
- You must describe your experience/results in a written report
- Details will follow later this semester

Course Material

The slides are the main course material

- Will be published on Toledo
- Optional information will also be put on Toledo

Any feedback is appreciated:

- Found a mistake in the slides? Let me know!
- General feedback is also welcome: complain during the course (not afterwards) so we can address it on time ☺

Exam format

Exam will be closed-book. Two types of questions:

- Questions to check whether you remember the main content of the course
- Questions about understanding and being able to reason about the content

Typical course advice: when studying slides ask yourself “what could they ask about this”?

Getting Help

- Ask questions during the lecutre!
  - Don’t wait with your questions until the end. Interaction is appreciated!
- For any other questions: send an e-mail
- An in-person meeting can be scheduled by e-mail
  - No “office hours” — this practice is uncommon here

Broader context

Other courses will explore certain aspects in more detail

- Advanced Methods for Security & Privacy by Design
- Development of Secure Software
- Capita Selecta Computer Science: Secure Software

The connections will be mentioned again in some lectures

- And feel free to ask/email questions about connections to other courses

Agenda for today

- Introduction to the Secure Development Lifecycle (SDL)
- Microsoft SDL for waterfall/spiral development
- Overview of other software security courses [CyberSecurity Basics]
- Introduction to threat modelling

Motivation recap: security is an issue!

Help Net Security
July 27, 2022
Share

Amazon's Twitch blames
data

The
data
time

INVESTMENT BANKING | LEGAL/REGULATORY

JPMorgan Chase Hacking Affects 76 Million Households

BY JESSICA SILVER-GREENBERG, MATTHEW GOLDSTEIN AND NICOLE PERLROTH OCTOBER 2, 2014 12:50 PM

528

for
data breach that hit millions

Attacks/Breaches | 4 MIN READ | ARTICLE

Blizzard Battle.net Security Breached, Passwords Accessed

ram,
ıbe User
n Massive
Data Leak

Motivation recap: old approach to security

Security & privacy ~~used to be~~ is often an afterthought

- Build the product and add security later.
- “We will add <product X> to secure it”.

But you can’t magically turn an insecure product into something secure

- You must think about security during the design and development of the product…

So, how do we secure things?

It’s not about using a particular tool or doing a specific task

Security must be an integral part of a product’s lifecycle

- Security is a process
- Security should, ideally, be testable
- Security should, ideally, be measurable

Security Development Lifecycle (SDL)

Synonyms

- Secure Software Life Cycle (SSLC)
- Security Development Lifecycle (SDL)
- Secure Development Lifecycle (SDL)
  - The term “Software Development Lifecycle” has same abbreviation…
- Secure Software Development Life Cycle (SSDLC)
- ~~Secure Development Life Cycle (SDLC)~~
  - Less common, conflicts with Software Development Life Cycle (SDLC)
- And more: Security by Design, Software Assurance,…

Hypothetical scenario

A company asks help on how to better secure their software

- Where to start? What to do? How to cover everything?

Possible answer: use a **Secure Development Lifecycle**!

Where to start?

Core idea is to secure your existing software lifecycle

- Don’t throw away the lifecycle you already have.
- Instead, **improve your existing lifecycle** step by step.

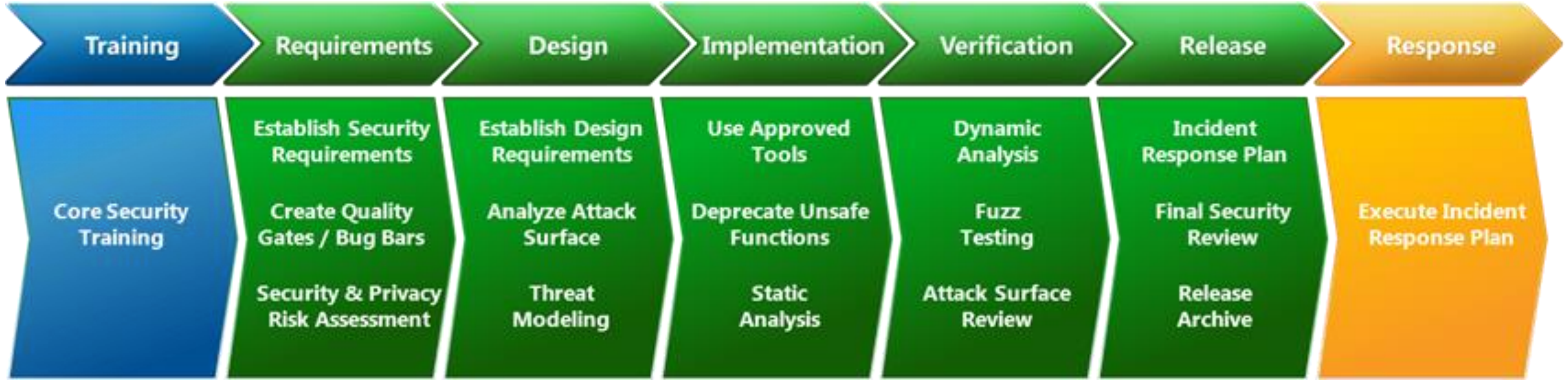
How to do this?

- Take a secure development lifecycle model as reference
    - This is essentially a collection of best practices
- Step by step **integrate security best practices** into the company’s existing software development lifecycle

The Microsoft SDL

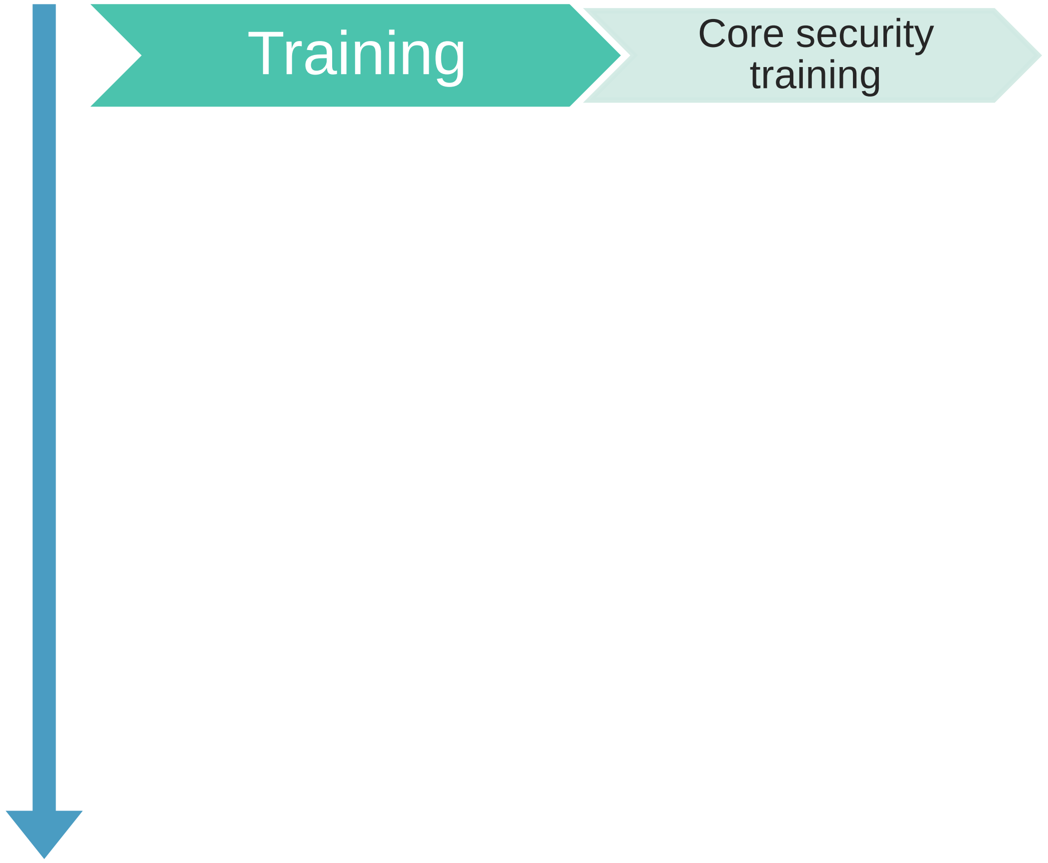
Microsoft SDL (old)

- Made public in 2008 (v3.2) with last update in 2012 (v5.2)
  - Replaced in late 2018 with a list of 12 best practices.*
- Around 160+ pages of information!
- The first public SDL. Was, and still is, very influential.

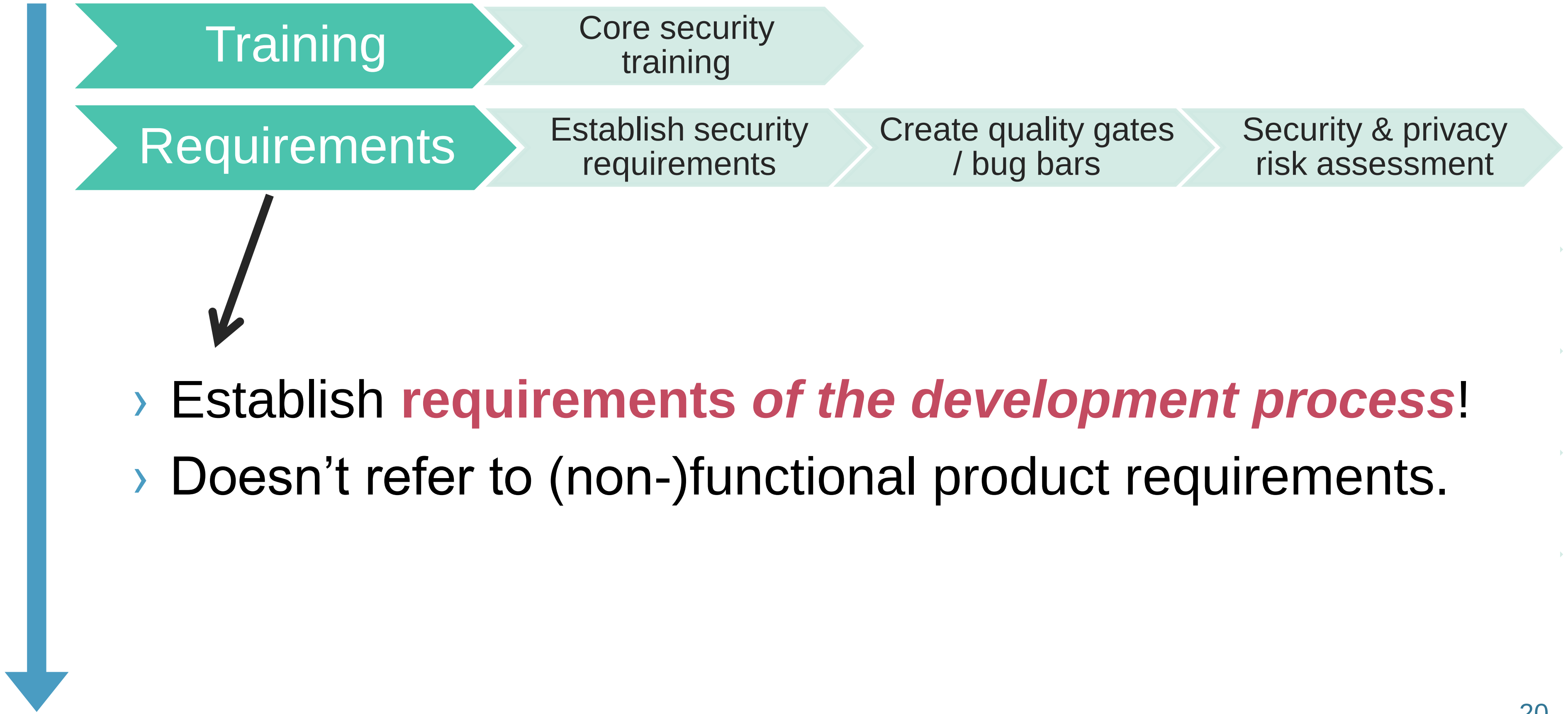


* Source: v5.2 SDL is under "Legacy archive" at https://www.microsoft.com/en-us/securityengineering/sdl/resources

Microsoft SDL (old) process model – version 5.2

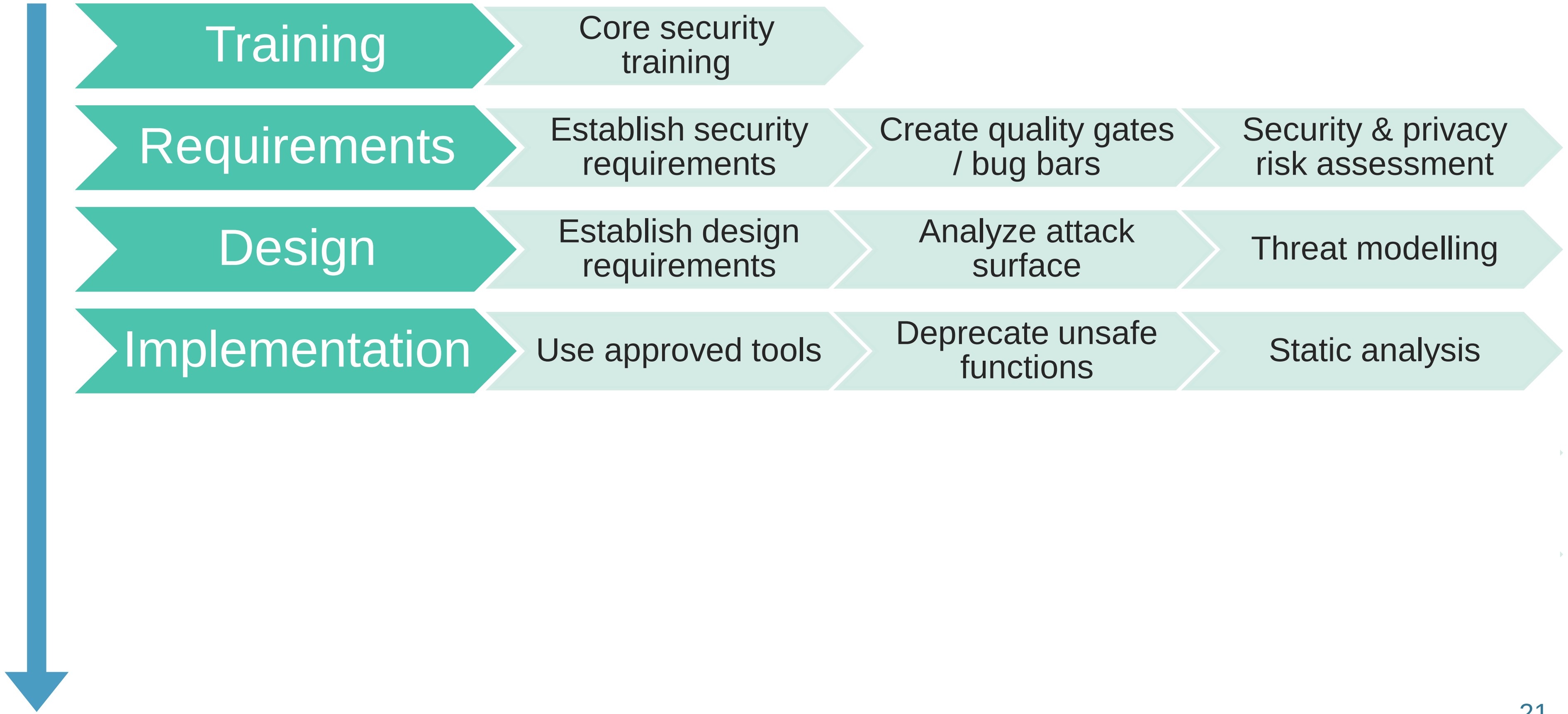


Microsoft SDL (old) process model – version 5.2


Training
Core security training
Requirements
Establish security requirements
Create quality gates / bug bars
Security & privacy risk assessment

- Establish **requirements *of the development process***!
- Doesn’t refer to (non-)functional product requirements.

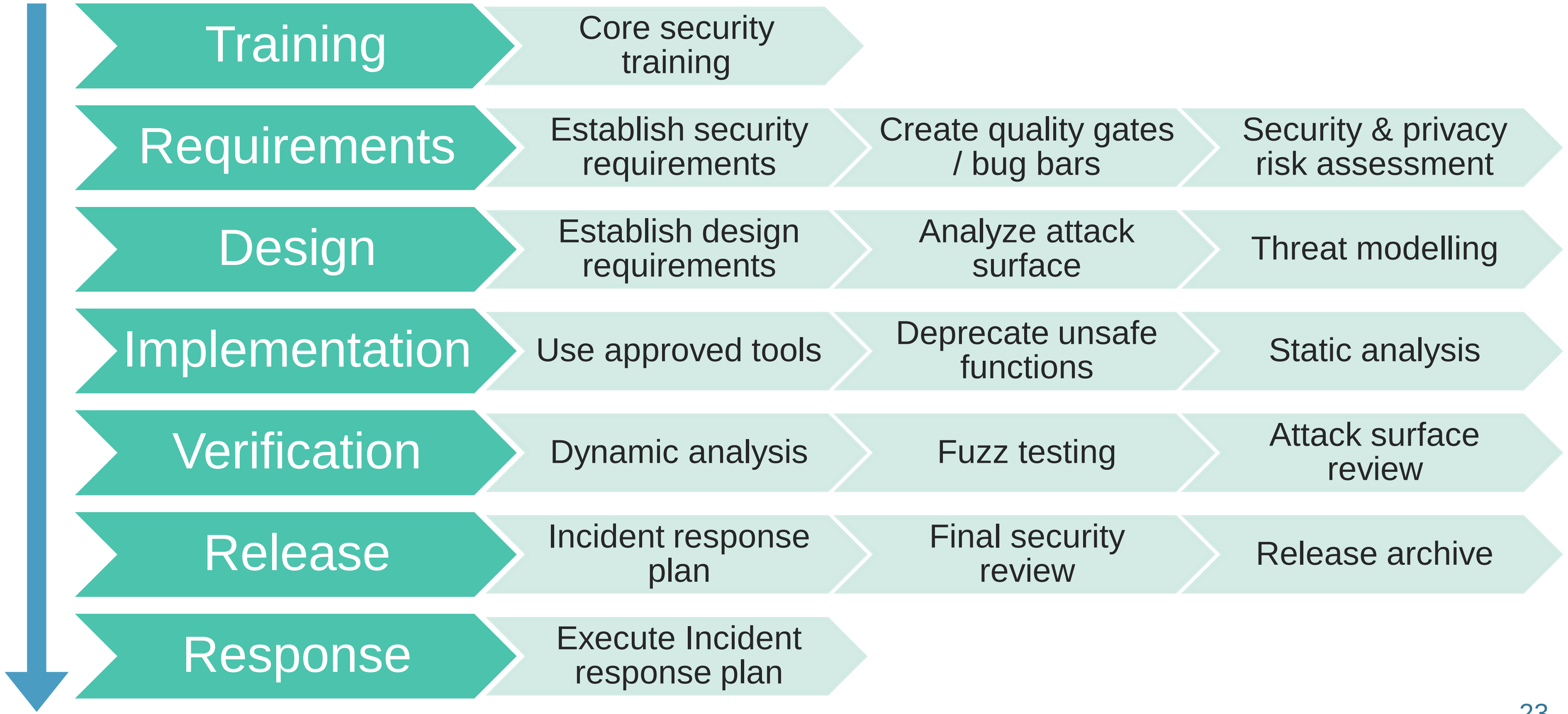
Microsoft SDL (old) process model – version 5.2


Training
Core security training
Requirements
Establish security requirements
Create quality gates / bug bars
Security & privacy risk assessment
Design
Establish design requirements
Analyze attack surface
Threat modelling
Implementation
Use approved tools
Deprecate unsafe functions
Static analysis

Microsoft SDL (old) process model – version 5.2

Training	Core security training		
Requirements	Establish security requirements	Create quality gates / bug bars	Security & privacy risk assessment
Design	Establish design requirements	Analyze attack surface	Threat modelling
Implementation	Use approved tools	Deprecate unsafe functions	Static analysis
Verification	Dynamic analysis	Fuzz testing	Attack surface review

Microsoft SDL (old) process model – version 5.2



Training	Core security training		
Requirements	Establish security requirements	Create quality gates / bug bars	Security & privacy risk assessment
Design	Establish design requirements	Analyze attack surface	Threat modelling
Implementation	Use approved tools	Deprecate unsafe functions	Static analysis
Verification	Dynamic analysis	Fuzz testing	Attack surface review
Release	Incident response plan	Final security review	Release archive
Response	Execute Incident response plan		

Microsoft SDL (old) background

- Based on the waterfall/spiral model
  - Waterfall: requirements → design → implementation → testing
  - Specifies **security & privacy best practices** to follow in each phase.
- Integrate practices step by step into your own model.
- Though deprecated, its best practices are still relevant today
  - For all the details, see the "SDL Process Guidance Version 5.2"
  - And see "Simplified Implementation of the Microsoft SDL"

Training phase

- All software developers must receive appropriate training
- First basic security training to cover fundamentals
  - Cover topics such as: secure design, threat modelling, secure coding, security testing, privacy
- If time and resources permit, train in advanced concepts
  - E.g., advanced security design, trusted UIs, custom threat mitigations

→ Cover **security basics and trends** in security & privacy

Requirements phase

- Develop and answer a short questionnaire to verify whether your development team is subject to SDL policies
  - If so, project must be assigned a **security advisor** (see next slides)
- Identify someone responsible for tracking & managing security for the product
  - Responsible for coordinating & communicating the status of security issues
  - Called the **security champion** (see next slides)
- Establish the security requirements (i.e., the development best practices we are about to see) for which the project will be held accountable.
  - = minimum security requirements
- Ensure that bug reporting tools can track security issues

Requirements (general)

Security / privacy advisor

- A subject-matter expert from **outside the project team**
- Provides project security & privacy oversight: can accept or reject security/privacy plans
- Acts as **auditor** during the development process and attest to successful completion of each security requirement

Requirements (general)

Security / privacy champion

- Expert(s) from the project that can negotiate and track minimum security and privacy requirements
- Responsible for **coordinating and tracking** security/privacy issues for the project.
- Also responsible for reporting status to the security advisor & to other relevant parties (e.g., development and test leads)

Requirements: quality gates & bug bars

Set minimum acceptable levels of security & privacy quality

- Enables teams to identify & fix security bugs during development
- **Quality gates**: predefined criteria that must be met before the project can proceed to the next phase
  - E.g., all compiler warnings must be triaged and fixed prior to code check-in
- **Bug bar: a quality gate** that applies to the **entire project**. It defines the severity thresholds of security vulnerabilities.
  - For example, no known vulnerabilities in the application with a “critical” or “important” rating at time of release
  - The bug bar, once set, should never be relaxed

Requirements: security & privacy risk assessment

Cost analysis:

- Before you invest time in design and implementation, determine if development and support **costs for (improving) security and privacy** are consistent with business needs
- Does it make sense for this project to spend this money?
- If the costs of security are high, the security level is lowered if possible. Or the project is killed if too costly.

Requirements: security assessment

To help estimate cost, identify aspects of the software that will require deep review:

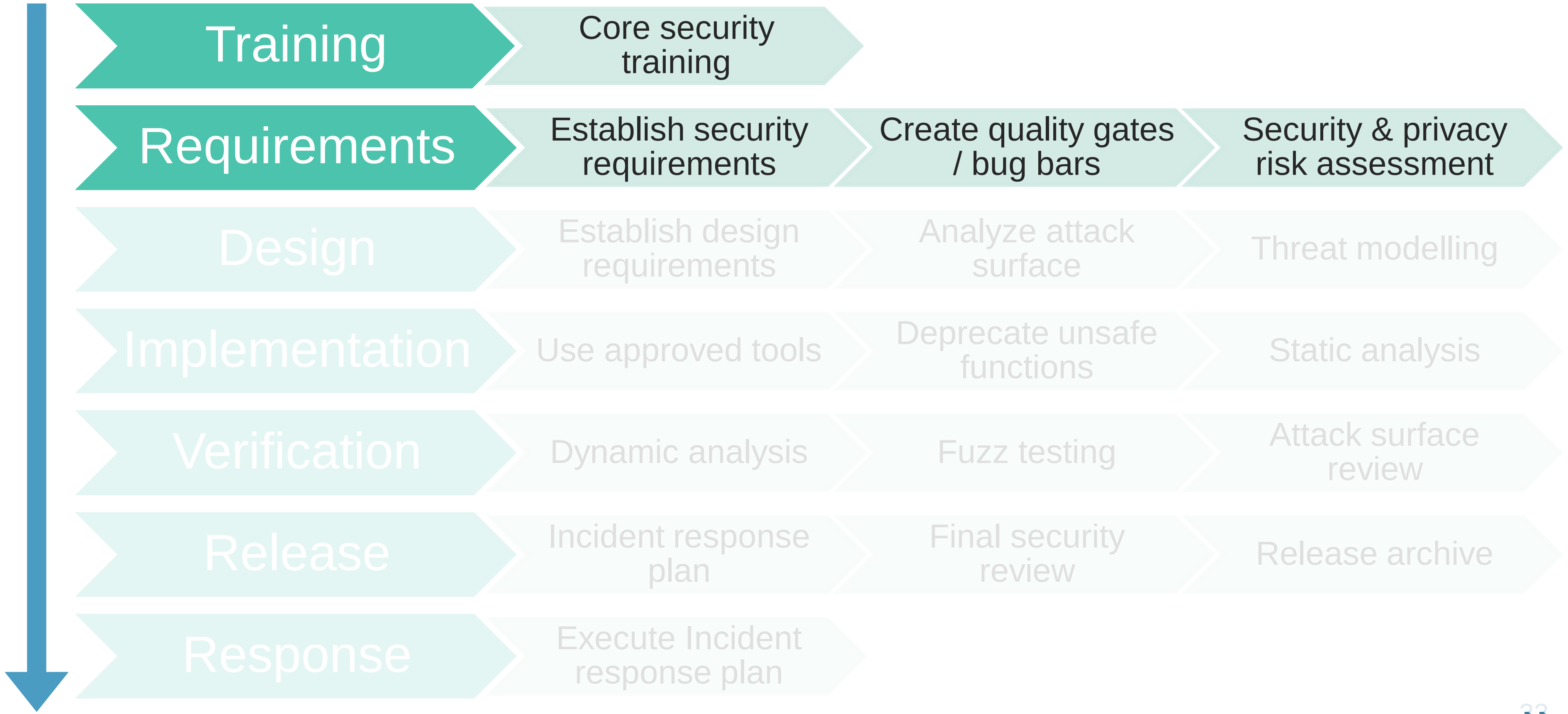
- Which parts of the project require **threat models** before release?
- Which parts of the project require **security design reviews**?
- Which portions require **penetration testing** by a third party?
- What is the specific scope of the **fuzz testing** requirements?

Requirements: privacy risk assessment

To help estimate cost, identify aspects of the software that will require deep review:

- What is the **privacy impact rating** of the application?
  - Low: no anonymous or personal data is transferred
  - Moderate: one-time user-initiated anonymous data transfer
  - High risk: stores/transmits Personal Identifiable Information (PII)
- Products with high privacy risk: before release, privacy advisor must certify the privacy requirements are satisfied

Microsoft SDL (old) process model – version 5.2



Microsoft SDL (old) process model – version 5.2

Training	Core security training		
Requirements	Establish security requirements	Create quality gates / bug bars	Security & privacy risk assessment
Design	Establish design requirements	Analyze attack surface	Threat modelling
Implementation	Use approved tools	Deprecate unsafe functions	Static analysis
Verification	Dynamic analysis	Fuzz testing	Attack surface review
Release	Incident response plan	Final security review	Release archive
Response	Execute Incident response plan		

Design: security requirements

Consider security early and don’t “bolt it on”. Follow best practices:

- Require secure **user authentication**
- Require **user consent** before high-risk privacy operations
- Use the principle of **least privilege** (e.g., UAC in Windows): privileged features of the app are separated out and placed behind an elevation prompt. This results in a reduced attack surface.
- All cryptography must comply with the MS **Crypto Standards**: key sizes, allowed algorithms, usage of TLS to send data, etc.
- Mitigate against **Cross-Site Scripting** (XSS) attacks.
- The design **must be reviewed** by a security advisor

Design: privacy requirements & recommendations

Consider privacy early! Follow best practices:

- If the project has a high privacy impact rating, identify a compliant design based on Microsoft Privacy Guidelines for Developing Software Products and Services
  - And ask your company's privacy expert for a "sanity check"
- For a project with a moderate to low privacy impact rating, it is recommended to follow the recommended guidelines

Design: recommended security practices

- **Reduce attack surface**: restrict access to system services, apply the principle of least privilege, employing layered defenses
- Examine past vulnerabilities and their root causes
- Deprecate outdated functionality
- Security review of sample source code released with the product
- Implement any new code using a memory-safe language
- Be careful with sensitive information in error messages
- Ensure appropriate logging is enabled
- And many more…

Design: risk analysis (threat modelling)

Risk analysis of the system is done

- What are remaining risks in the design? Identify them and address them now during the design phase.
- Use Microsoft STRIDE methodology for threat modelling.
  - Each threat model must include diagrams that show the software and all trust boundaries
  - The STRIDE method will be explained in detail later
  - Produce mitigations for all threats
- Verify that threat models exist for all attack surfaces

Design: security feature vs secure feature

- **Security feature**: program functionality with security implications, such as authentication, encryption, a firewall.
- **Secure feature**: features whose functionality is well engineered with respect to security (proper input validation, usage of well-known crypto libraries, etc.)

A “security feature” is not necessarily a “secure feature”

- Even the most secure design is rendered pointless by a low quality and insecure implementation.
- You can’t just “bolt on” security. Must be done carefully.

Implementation: security requirements

- Create documentation: provide enough info so users can make informed decisions on how to use a program securely
- Establish and follow best practices for development
  - Use HTTPOnly/secure cookies, enable low-level attack mitigations
  - Use static code analysis, don’t use banned/deprecated functions
  - Use secure methods to access database (prevent SQL injection)
  - Load DLLS securely, mitigate cross-site request forgery (CSRF)

→ In general, review available information and define, communicate, and document all best practices and policies

Implementation: security recommendations

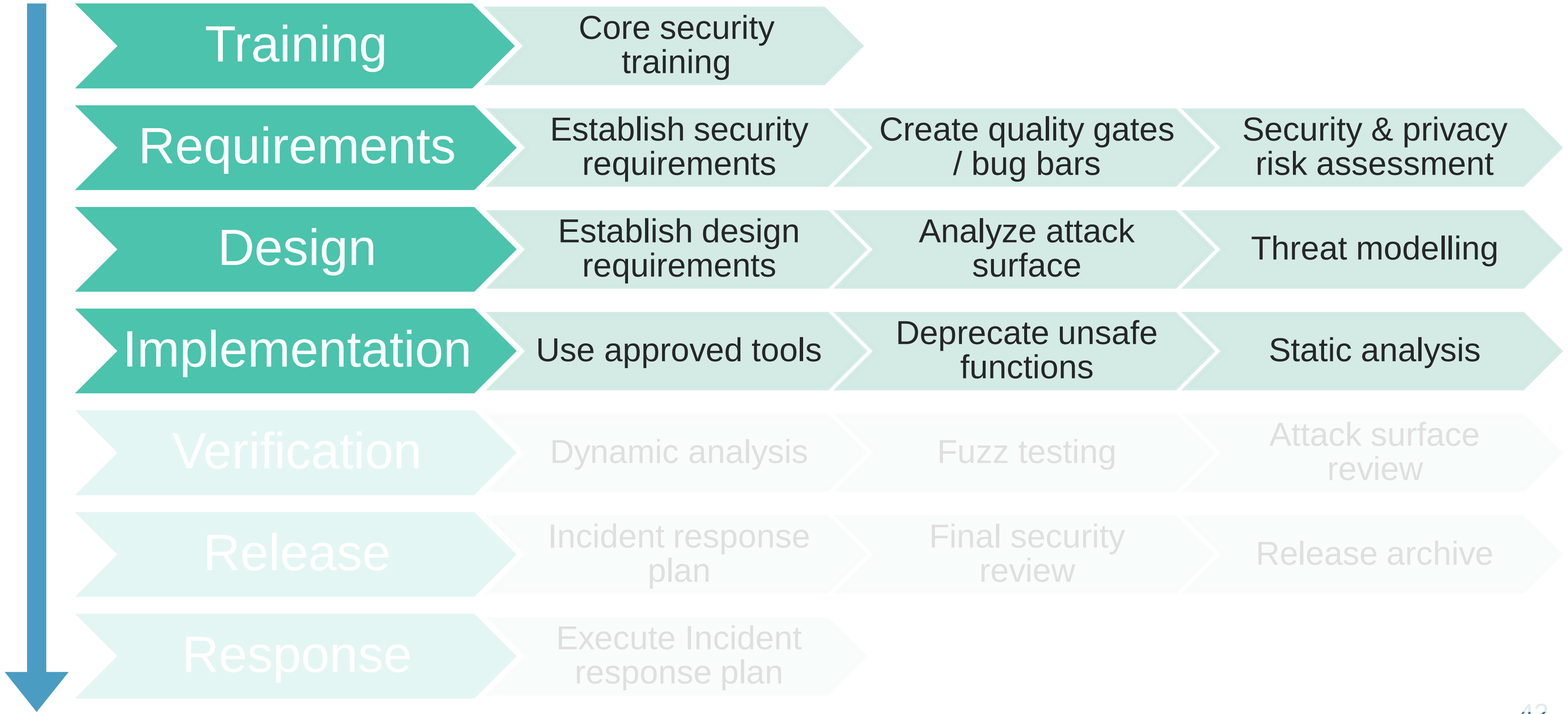
Microsoft also recommends several best practices:

- Avoid using `eval()`, defend against clickjacking attacks
- Set free'd points to NULL:

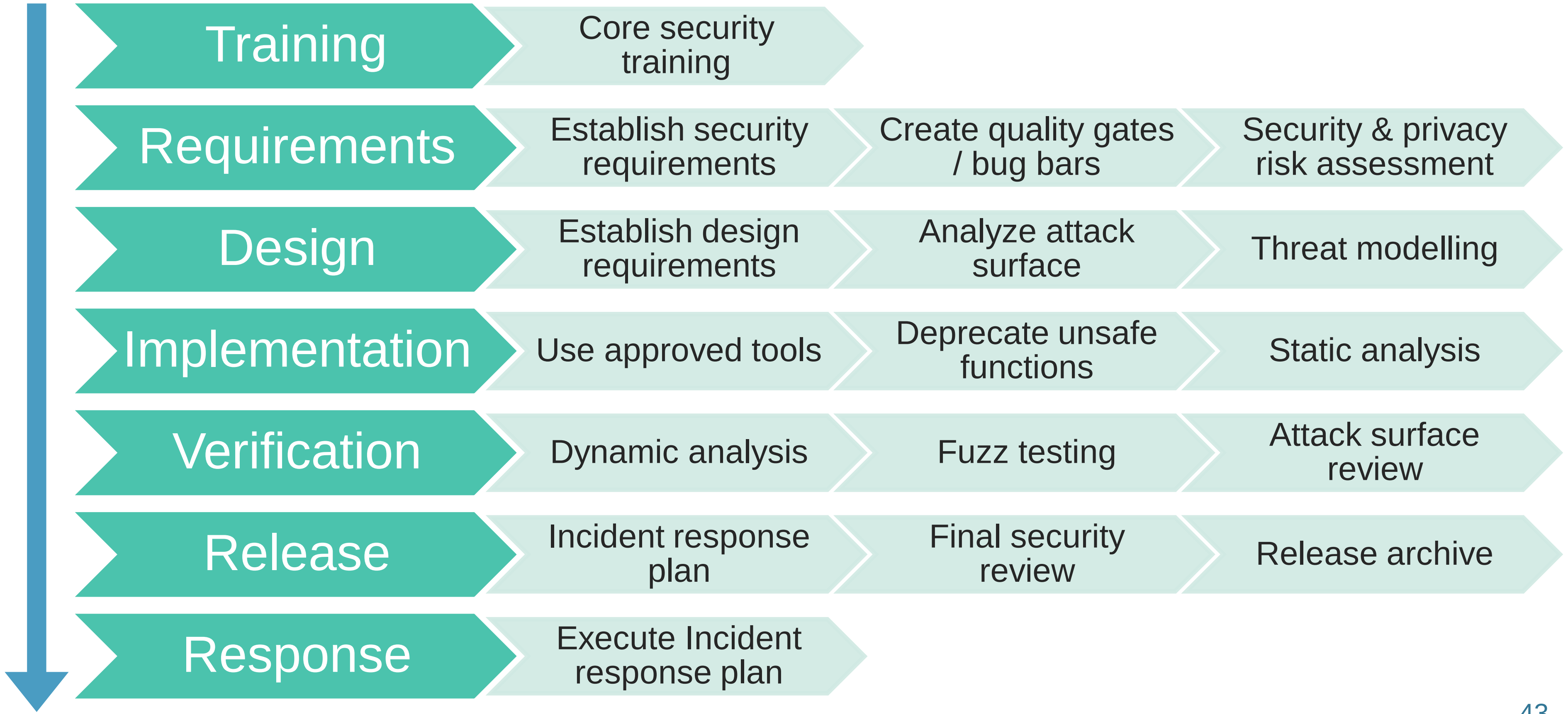
```
char *p = new char[N];
...
delete [] p;
// Add this statement after the delete operator:
p = NULL;
```

- And several more…

Microsoft SDL (old) process model – version 5.2



Microsoft SDL (old) process model – version 5.2



Training	Core security training		
Requirements	Establish security requirements	Create quality gates / bug bars	Security & privacy risk assessment
Design	Establish design requirements	Analyze attack surface	Threat modelling
Implementation	Use approved tools	Deprecate unsafe functions	Static analysis
Verification	Dynamic analysis	Fuzz testing	Attack surface review
Release	Incident response plan	Final security review	Release archive
Response	Execute Incident response plan		

Verification (aka testing)

Perform dynamic program analysis (part one):

- Use tools that monitor application behavior for memory corruption, user privilege issues, and other security problems
- For example, use AppVerifier on Windows platforms:
  - Checks that apps don't hide memory access violations by using structured exception handling
  - Tests to ensure that invalid handles are not being used
  - Checks for memory corruption issues on the heap
  - Checks if the app can also run in an environment with less privileges
  - And so on…

All AppVerifier checks: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/application-verifier

Verification (aka testing)

Perform dynamic program analysis (part two):

- Use approved cross-site scripting scanning test tools
- Use approved scanner to check for XML parsing problems
- Use the “Device Path Exerciser” on kernel-mode drivers
  - Calls drivers through a variety of user-mode I/O interfaces with valid, invalid, meaningless, and poorly-formatted buffers and data.
  - Can reveal improper driver design or implementation that might result in system crashes or make the system vulnerable.

Device Path Exerciser: http://msdn.microsoft.com/en-gb/library/ff544851.aspx

Verification (aka testing)

Perform fuzzing

- A specialized form of dynamic analysis where the program is fed malformed or random input to try to crash it
- Required where input to file parsing code could have crossed a trust boundary
- Required to fuzz remote procedure call (RPC) interfaces
- Network fuzzing: “each network parser must successfully handle 100,000 malformed packets without error”
  - These days we can test using a lot more malformed packets ☺

Verification (aka testing)

Security push:

- Time is reserved to **work only on security** (e.g., 2-3 weeks)
- Assures every project has enough time to work on security
- Consists of several tasks: threat model updates, code review, testing, and thorough documentation review
- Re-evaluate the attack surface of the software:
  - Reveals which components are the highest risk areas
  - Address new issues: prolong security push, disable component by default, don’t ship component, deprecate component, etc.

Release

- Privacy review: check changes in the language of a notice, collection of different data, etc.
- Create an incident response plan
  - Establish contact info for people who respond to security incidents
  - Establish sustainable model to release security patches
  - Response process for servicing code that has been licensed from third parties
  - Processes must be in place to quickly react to disclosed vulnerabilities

Release: Final Security Review (FSR)

- Examine threat models, exception requests, tool output, and check determined quality gates or bug bars
- Possible outcomes:
  - passed FSR
  - passed FSR with exceptions: remaining issues are logged and corrected in the next release
  - FSR escalation: remaining issues must be addressed or escalated to execute management for a decision
- Not a moment to perform security activities that were previously ignored or forgotten!

Release and archive

- Security advisor must **certify** (using the FSR and other data) that the project team has satisfied security requirements
- Privacy advisor must **certify** that the project team has satisfied the privacy requirements
- All **information and data must be archived** to allow for post-release servicing of the software
  - Includes: specifications, source code, binaries, private symbols, threat models, documentation, emergency response plans, license and servicing terms for any third-party software.

Response

Execute incident response plan

- Assure you respond appropriately to vulnerability reports
- Detect attempted exploitation of those vulnerabilities

Recap on Microsoft SDL (old)

- Existed for a long time. Now replaced by 12 best practices.
- A lot of documentation is online
- Oriented towards software vendors
- Supporting tools also exist for: threat modelling, coding guidelines, etc.
- Remember: no single model will fit your organization. Take inspiration from several models and adapt them.

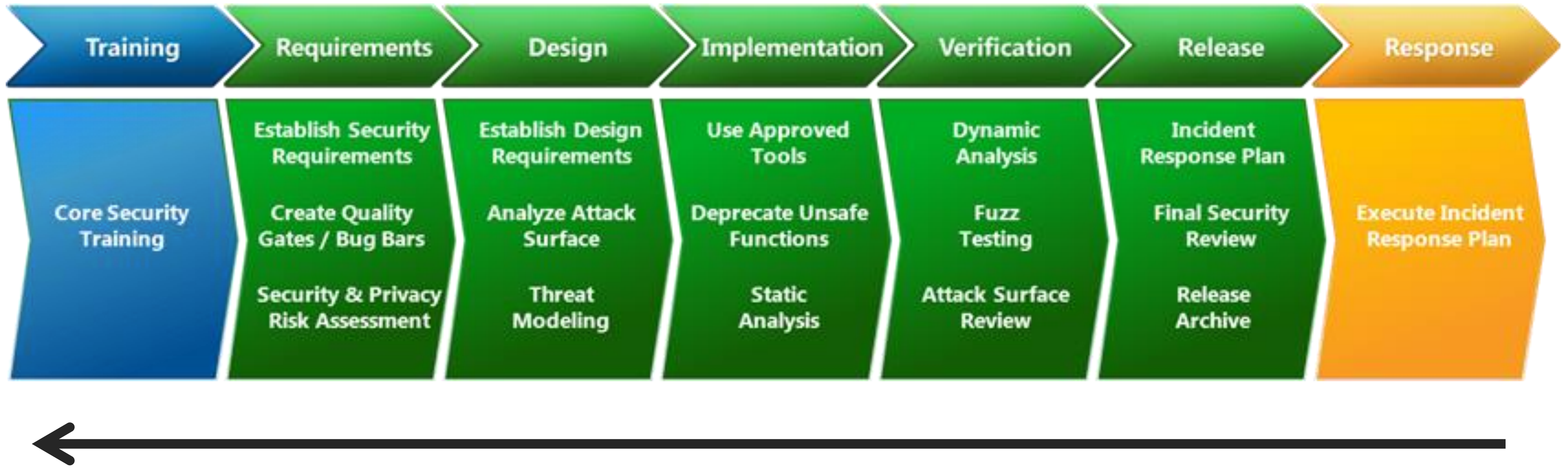
Impact of using the Microsoft SDL

- It’s quite an amount of time to invest!
- Estimated to cost 5% to 15% of your project budget*
  - Depending on how risky the application
- How to convince a company of adopting an SDL?
  - See it as a **competitive advantage** (our product is more secure)
  - **Compliance** with regulations (GDPR, PCI, etc.)
  - Detecting and fixing bugs early **costs a lot less than fixing** them late in the development lifecycle

* Source: “SDLC: Introduction and process models” by Bart De Win at SecAppDev, 2017.

Summary

- Security must be considered in every phase of the lifecycle
- Must start with security early, called the **shift left trend**:



Shift-left: assuring security early in the software lifecycle

Other courses in the Software Track

Security Through the Software Lifecycle

- Overview of other best practices when writing software
  - Secure designs, writing secure code, secure build pipeline, etc.
- Strong focus on testing methods to detect vulnerabilities
  - For instance, fuzzing and/or symbolic execution
  - With a focus on detecting vulnerabilities in network services
  - There will be a hands-on **project about testing/fuzzing**!
- Overview of pentesting techniques and tools (web & mobile)
- Post-market aspects: incidence response, bug bounties, etc.

Advanced Methods for Security & Privacy by Design

State-of-the art methods, often based on the latest research

- (Advanced) threat modelling techniques
- GDPR compliance analysis
- Vulnerability management

Novel and research-oriented topics in the domain of security and privacy modelling.

- Precise focus points may change yearly

Capita Selecta Computer Science: Secure Software

Advanced & current themes in software security

- Often, but not necessarily, research-oriented
- May contain projects that you work on

Example previous focus topics:

- Formal verification of programs
- Using SGX to create secure modules
- “Hands-on-hacking” modules to learn about exploits
- …

Threat modelling

How to identify threats to defend against?

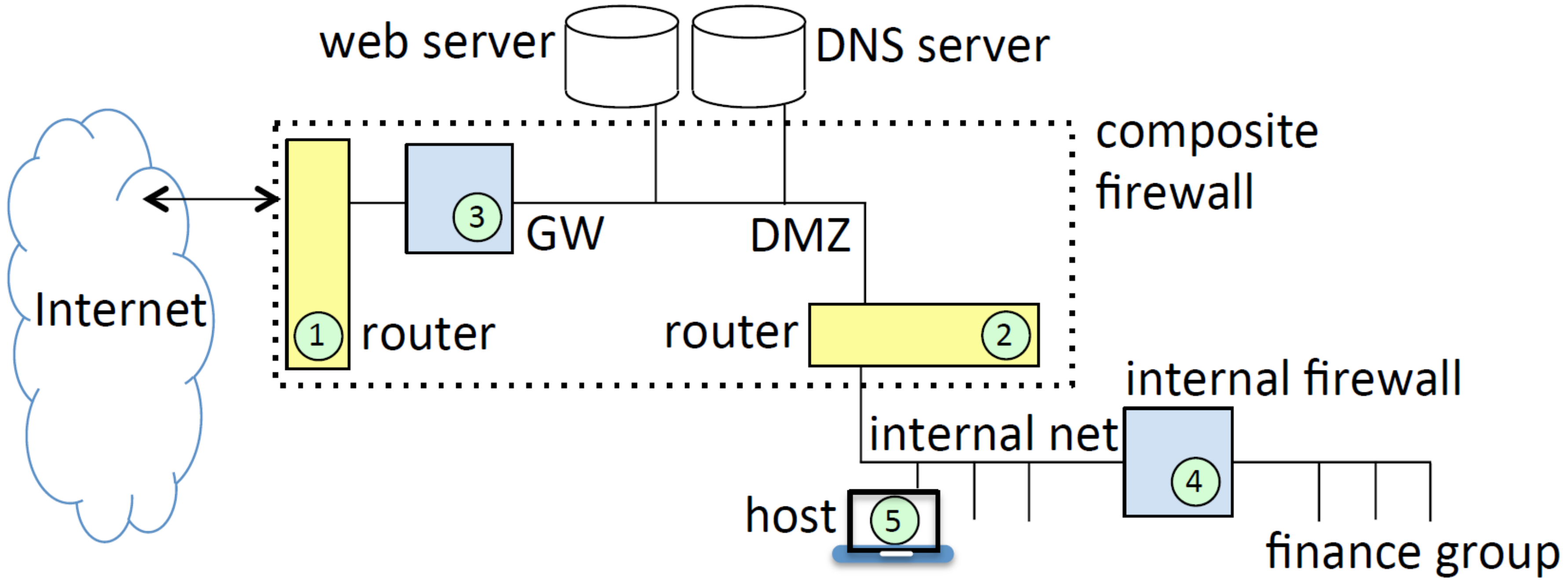
Use threat modelling!

- A method to identify threats, threat agents, & attack vectors
- Many threat modelling methods exists.

Today we briefly introduce **diagram-driven** threat modelling

- Start with an architectural representation of the system
- = a diagram containing all components of the system, and all communication and data flows between the components
- See section 1.5.1 of Computer Security and the Internet

Example diagram



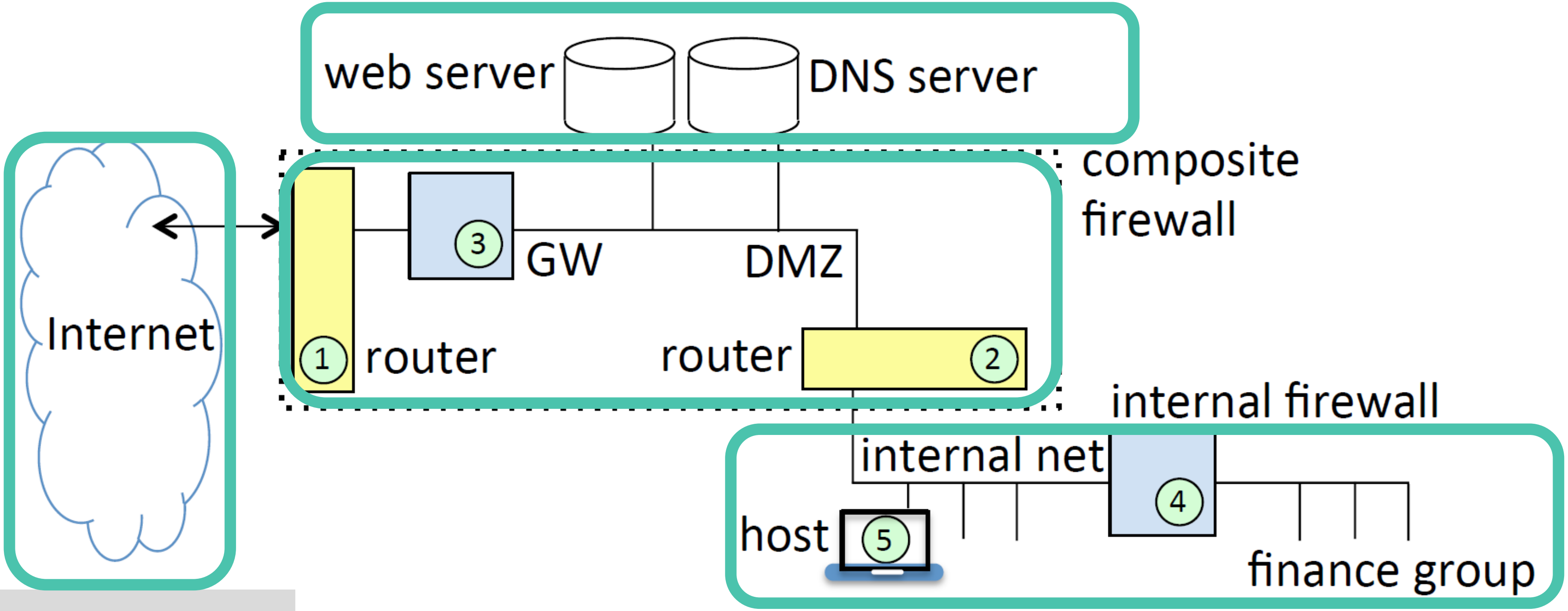
Source: **Computer Security and the Internet:** *Tools and Jewels from Malware to Bitcoin*, Second Edition by Paul C. van Oorschot. Springer, 2021.

Diagram-driven threat modelling

1. Identify and mark **system gateways** where system controls restrict or filter communications.
2. Use these gateways to create **trust domains**. For instance, users behind a firewall have different trust assumptions associated with them.

Example diagram

DMZ



Untrusted internet

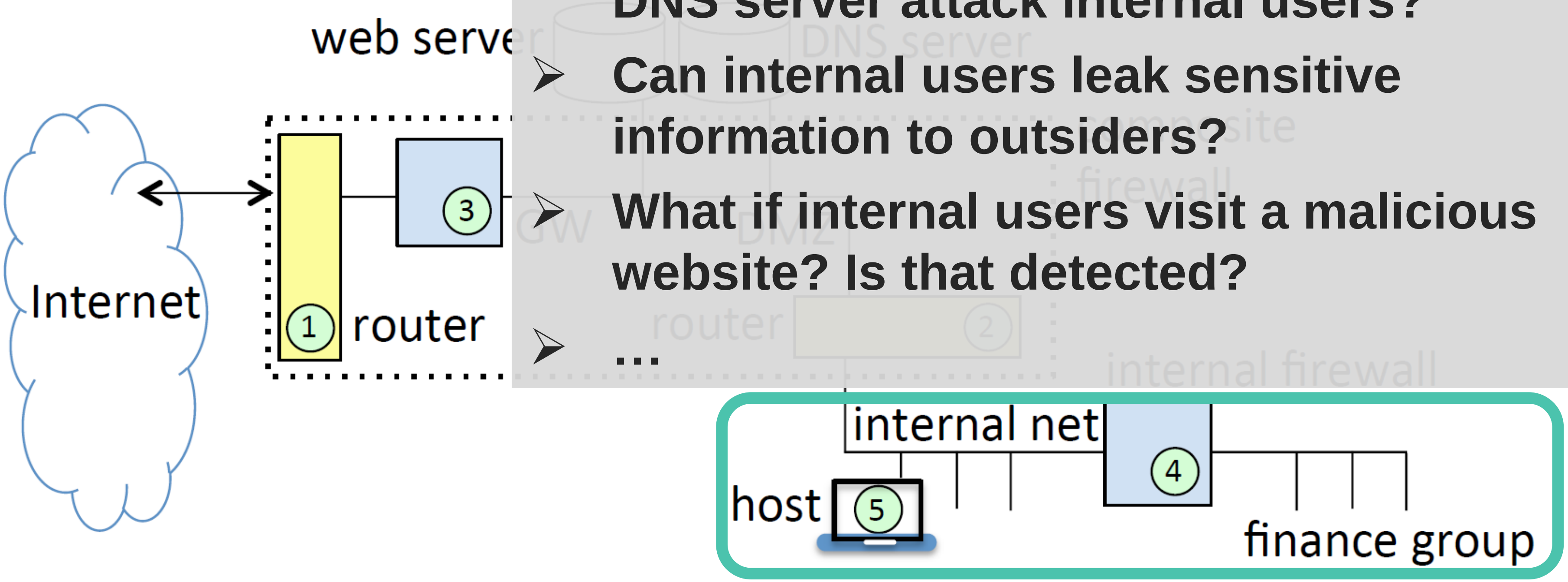
Internal = trusted users

Source: **Computer Security and the Internet:** *Tools and Jewels from Malware to Bitcoin,* Second Edition by Paul C. van Oorschot. Springer, 2021.

Diagram-driven threat modelling

1. Identify and mark **system gateways** where system controls restrict or filter communications.
2. Use these gateways to create **trust domains**. For instance, users behind a firewall have different trust assumptions associated with them.
3. Now ask "**where can bad things happen?** How?"
    - ➢ Focus on each component, link, and trust domain
    - ➢ How might your trust assumptions, or expectations of who controls what, be violated?

Example diagram



- Can internal users attack each other?
- Can a compromised web server or DNS server attack internal users?
- Can internal users leak sensitive information to outsiders?
- What if internal users visit a malicious website? Is that detected?
- …

Internal = trusted users

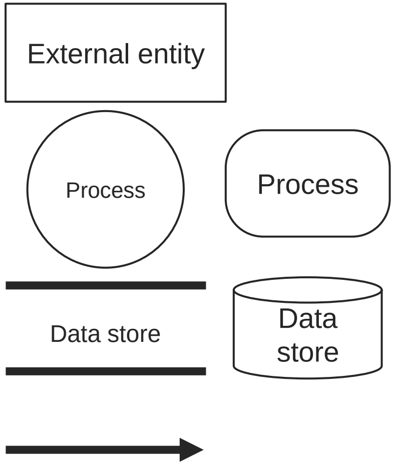
Source: **Computer Security and the Internet:** *Tools and Jewels from Malware to Bitcoin*, Second Edition by Paul C. van Oorschot. Springer, 2021.

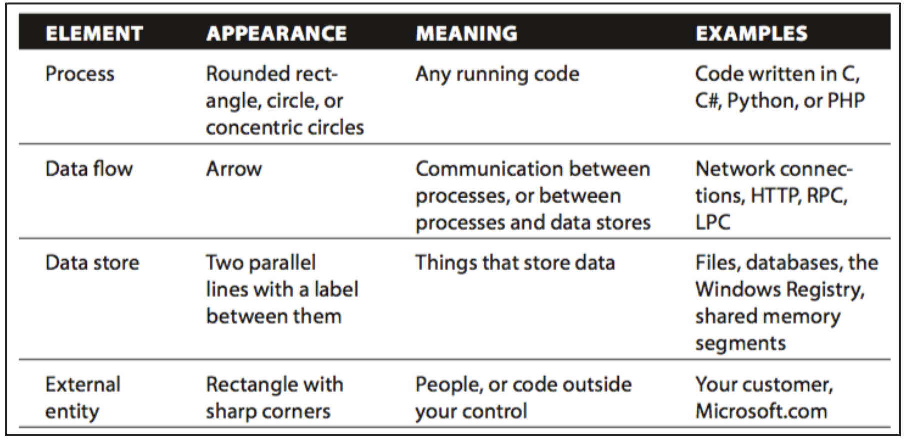
Diagram-driven threat modelling

- Trace the **flow of data** through the system for a simple task, transaction, or service. Again ask: “What could go wrong?”
- Trace through **user actions** from the time a task begins until it ends. Again ask: “What could go wrong?”
- Revisit your diagram and highlight where sensitive **data files** are stored—on servers, user devices? Assure all access **paths to this data** are shown.
- Might attackers access or **alter data that is sent** over any of the links?

Can also use a data flow diagram (DFD)

- External entity
- Process
- Data store
- Data flow





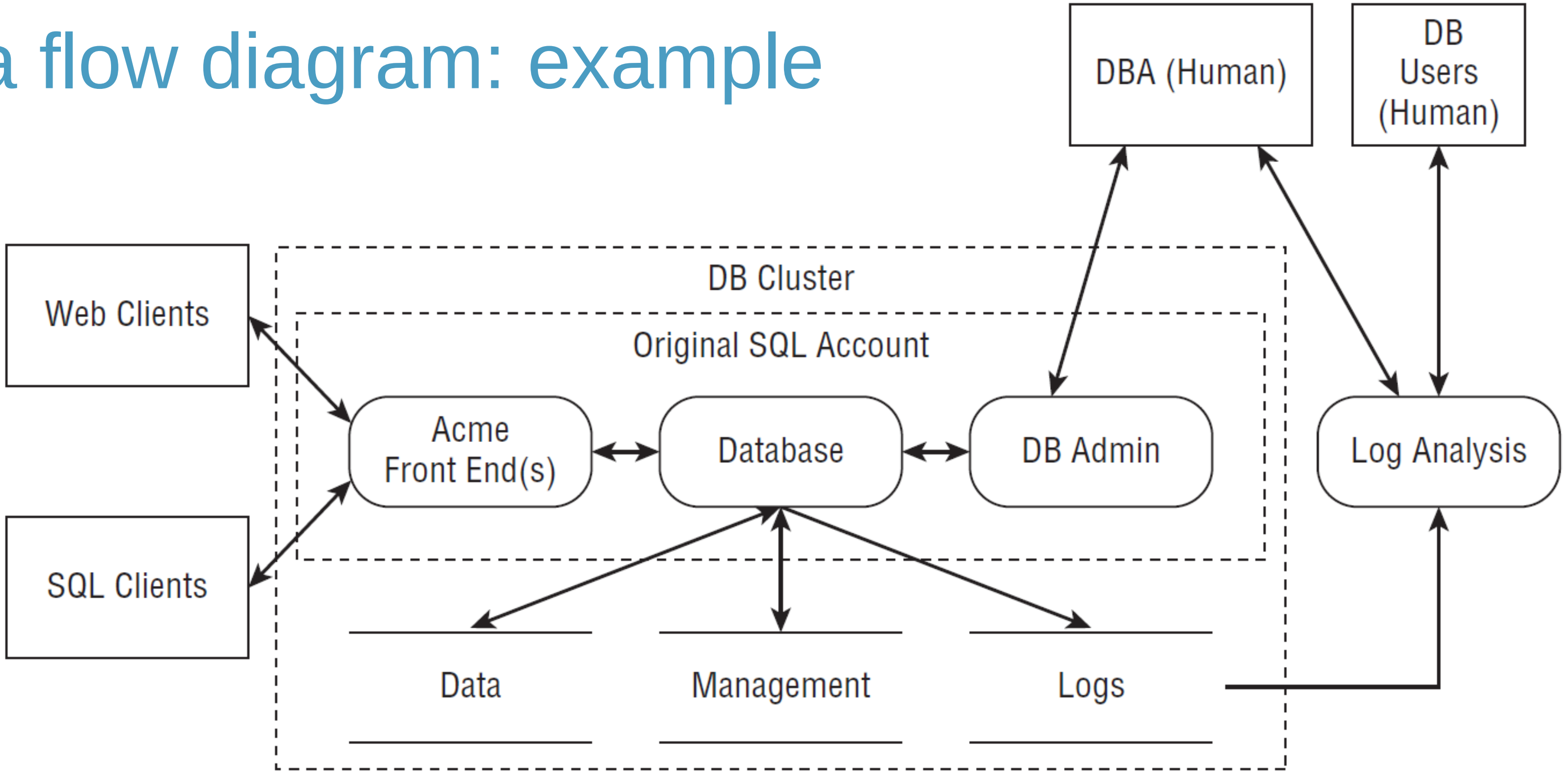
ELEMENT	APPEARANCE	MEANING	EXAMPLES
Process	Rounded rect-angle, circle, or concentric circles	Any running code	Code written in C, C#, Python, or PHP
Data flow	Arrow	Communication between processes, or between processes and data stores	Network connec-tions, HTTP, RPC, LPC
Data store	Two parallel lines with a label between them	Things that store data	Files, databases, the Windows Registry, shared memory segments
External entity	Rectangle with sharp corners	People, or code outside your control	Your customer, Microsoft.com

Shostack, A., 2014. Threat Modeling. Wiley.

- Trust boundary = a place where principals (with different privileges) interact

→ Use resulting data flow diagram for threat modelling

Data flow diagram: example



Use methods such as Microsoft STRIDE to go over everything that could go wrong (other courses cover the details).

Identify threats: Microsoft STRIDE

- **Spoofing**: attacker impersonates an entity/process in the system.
- **Tampering**: attacker modifies data/code.
- **Repudiation**: attacker can deny having performed an (illegal) action.
- **Information disclosure**: attacker can access info which should be inaccessible.
- **Denial of Service**: attacker can prevent the service from being available.
- **Elevation of privileges**: attacker (or valid user) gains more privileges than anticipated.

References

Required reading:

- Simplified Implementation of the Microsoft SDL, 2012.
- Section 1.5.1 of Computer Security and the Internet for a short 2-page introduction to diagram-based threat modelling

Optional reading:

- Microsoft Security Development Lifecycle (SDL) Process Guidance - Version 5.2 Detailed explanation of the SDL that was covered in this lecture (including more examples).
- The Security Development Lifecycle book by Michael Howard and Steve Lipner, 2006.