COMET

Competence Centers for Excellent Technologies

Join our security Meetup!

Software Security 101

Secure Coding Basics

sec4dev, Feb 23, 2021

Thomas Konrad, SBA Research

Bundesministerium Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie
Bundesministerium Digitalisierung und Wirtschaftsstandort
FFG Forschung wirkt.
wirtschafts agentur wien Ein Fonds der Stadt Wien
European Commission
FWF Der Wissenschaftsfonds.
netidee OPEN INNOVATIONS
DEPARTMENT OF COMMERCE UNITED STATES OF AMERICA

Photo by Brian Wangenheim on Unsplash

```
$ whoami
Thomas Konrad
$ id
uid=123(tom)
gid=0(SBA Research)
Gid=1(Vienna, Austria)
gid=2(Software Security)
gid=3(Penetration Testing)
gid=4(Software Development)
gid=5(Security Training)
gid=6(sec4dev Conference & Bootcamp)
```

Agenda

1. Introduction
2. Secure coding practices
3. Clean code
4. Secure SDLC fundamentals
5. Dependency management
6. Common vulnerability classes
7. Learning resources

Ask Questions on Slido!

https://sli.do – **#sec4dev** – Room „Software Security 101: Secure Coding Basics“

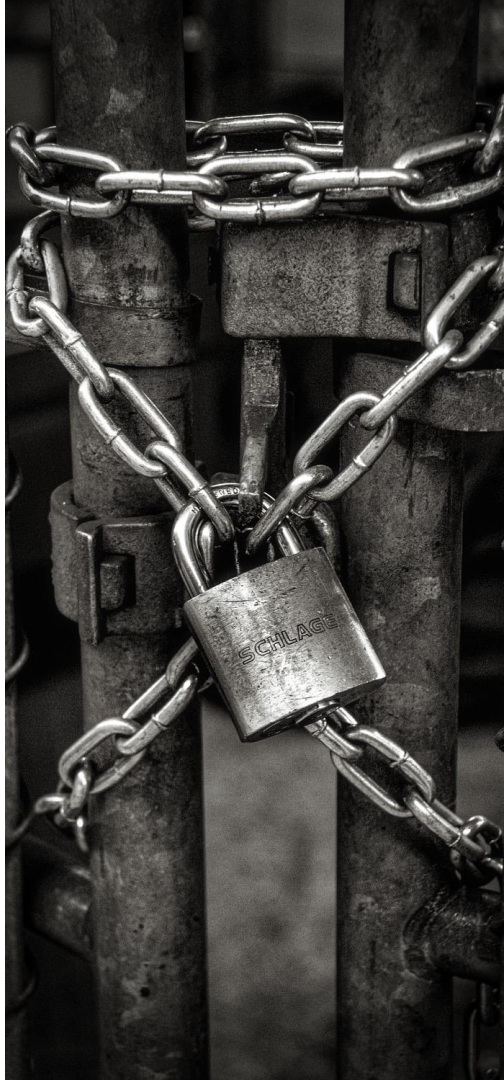

Let’s navigate the
software security landscape
together

Introduction

Why we are here, security principles and criteria for choosing a language

Why Are We Here?

- Customer expectation
- Company expectation
- Compliance
- Intrinsic motivation
- Imposed security posture



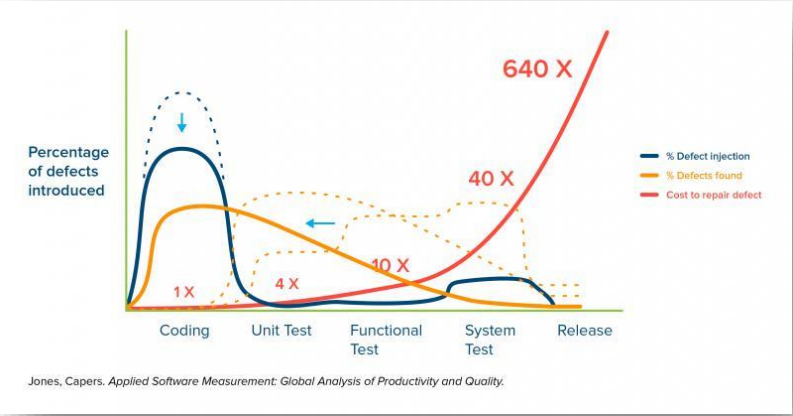
SBA Research, 2020

Software Is Everywhere

- Many companies are software companies, if they realize it or not
- Highly connected products open a myriad of attack vectors
- Healthy growth is only possible with security as a first-class citizen

Security and Quality

- Secure software is typically high-quality software
- Security as a usual quality requirement, not something “on top”
- Most cost-effective in the long term when considered from the start



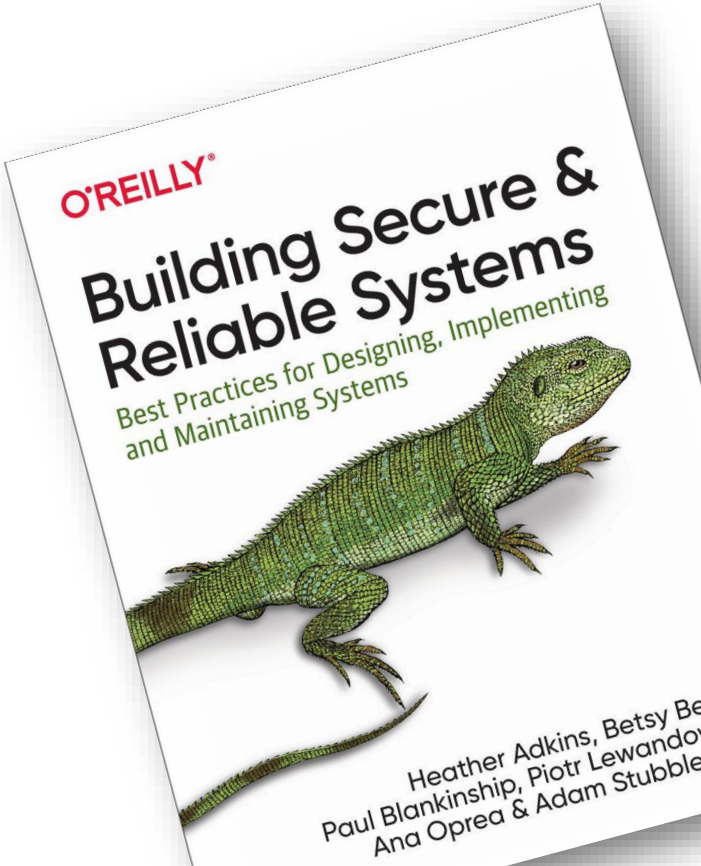
Technical Debt

"Weeks of coding can save you hours of planning"

Initial Velocity vs. Sustained Velocity

- *„We'll add security later"* –
  No, you won't.
- You hope to gain initial velocity
- But you'll lose sustained velocity

Book recommendation: "Building Secure and Reliable Systems" by Heather Adkins et. al.



Flaw vs. Bug

https://www.britannica.com/topic/Tacoma-Narrows-Bridge

Security Principles

- **Core Security Concepts**
  - Confidentiality
  - Integrity
  - Availability

  - Authentication
  - Authorization
  - Accountability

Image source: https://www.technologygee.com/confidentiality-integrity-availability-concerns-comptia-it-fundamentals-fc0-u61-6-1/

Security Principles

- **Design Security Concepts**
  - Least Privilege
  - Separation of Duties
  - Defense in Depth
  - Fail Secure vs. Fail Safe
  - Economy of Mechanisms
  - Complete Mediation
  - Open Design
  - Least Common Mechanisms
  - Psychological Acceptability
  - Weakest Link
  - Leveraging Existing Components

Security Criteria for Choosing a Language

I'll tell you a secret!

- Some languages protect against certain vulnerability classes by design
- However, secure software *can* be written in *any* language
- Mastering the language means mastering security

Security Criteria for Choosing a Language

But why is there so much low-quality code in specific languages?

- Some languages have very low entry barriers
- There will also be less skilled people writing and publishing code
- But that does not mean the language is bad!
- We *need* languages with low entry barriers!

Security Criteria for Choosing a Language

- Memory safety
- Type safety
- (Parallelization support)
- Sandbox support
- Availability of secure frameworks

Memory Safety

Memory safety has many flavors

- Array bounds checks
- Pointer arithmetic
- Null pointers
- Accessibility of unallocated, de-allocated, or uninitialized memory

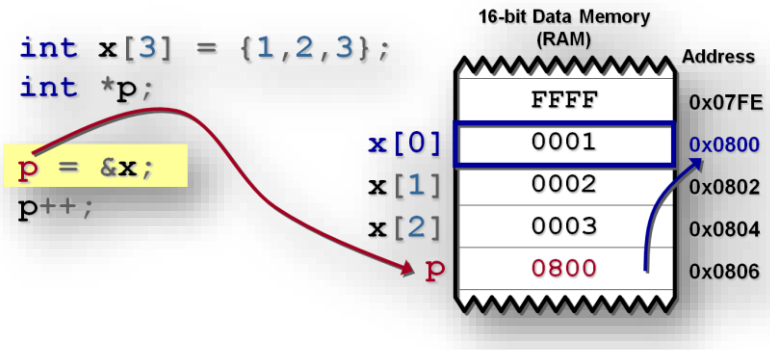


Image source: https://microchipdeveloper.com/tls2101:pointer-arithmetic

Memory Safety: Why Bother?

Non-memory-safe languages are susceptible to some vulnerability classes by design

- Buffer overflows
- Heap overflows
- Memory leaks

Memory Safety

- **Languages with no memory safety**
  - C
  - C++
  - Machine code
- **Languages with some form of memory safety**
  - Java
  - C#
  - Rust (mostly)
  - Go (mostly)
  - PHP
  - Python
  - Ruby
  - …

Type Safety

What is it?

- E.g., „this variable holds an integer“ or „this array has 10 elements“
- Type checking can happen at compile time or at runtime
- Type safety means if assertions are guaranteed at runtime

Type Safety

PHP non-type-safe example

```
$a = "42"; // now $a is a string
$a = $a + 42; // now $a is an integer
$a = $a + 23; // now $a is still an integer
$a = $a + 1.3; // now $a is a float
```

Type Safety

JavaScript non-type-safe example

```
> '5' - 3
2         // weak typing + implicit conversions * headaches
> '5' + 3
'53'      // Because we all love consistency
> '5' - '4'
1         // string - string * integer. What?
> '5' + + '5'
'55'
> 'foo' + + 'foo'
'fooNaN' // Marvelous.
> '5' + - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - + - - - '-2'
'52'      // Apparently it's ok

> var x * 3;
> '5' + x - x
50
> '5' - x + x
5       // Because fuck math
```

https://www.reddit.com/r/ProgrammerHumor/comments/2ubhql/please_dont_hate_me_javascript_devs/

Type Safety: Why Bother?

Why should we care?

- Type safety has long-term advantages
- Better IDE support (type hints)
- Better tool support (SAST)
- Less unexpected errors

Image source: https://www.jetbrains.com/de-de/idea/

Type Safety

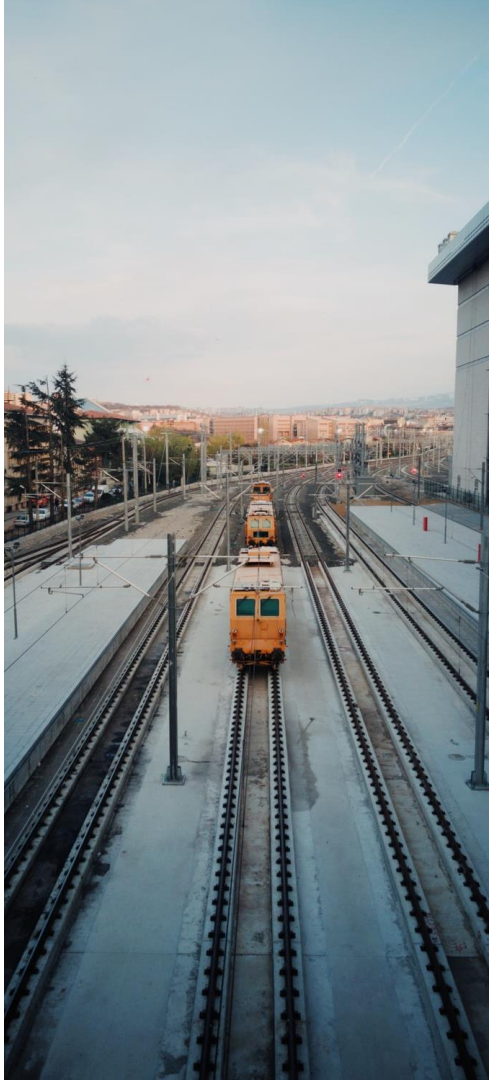
- **Languages with no type safety**
  - JavaScript
  - TypeScript (!)
  - PHP (but PHP is moving towards type safety)
  - Python
- **Languages with some form of type safety**
  - Java
  - C#
  - Rust
  - Go
  - C
  - C++

Parallelization Support (Advanced)

This is an advanced topic!

- Some languages are designed for robust parallel computing (Clojure, Elixir, Erlang, Haskell, Rust, …)
- Others have less focus on parallelization

Inform yourself before you start!



Sandbox Support

Suppose there will be vulnerabilities!

- Attack surface reduction is key to a sound security architecture
- Lock each process down to only the necessary capabilities
- Sandbox technology can help

Sandbox Support

- **Operating system level**
  - AppArmor
  - SELinux
  - seccomp
  - Chroot
- **Platform level**
  - Your web browser!
- **Language level**
  - .NET Code Access Security (CAS)
  - Java Security Manager

Related sec4dev Talk!

seccomp For Developers - Writing More Secure Applications

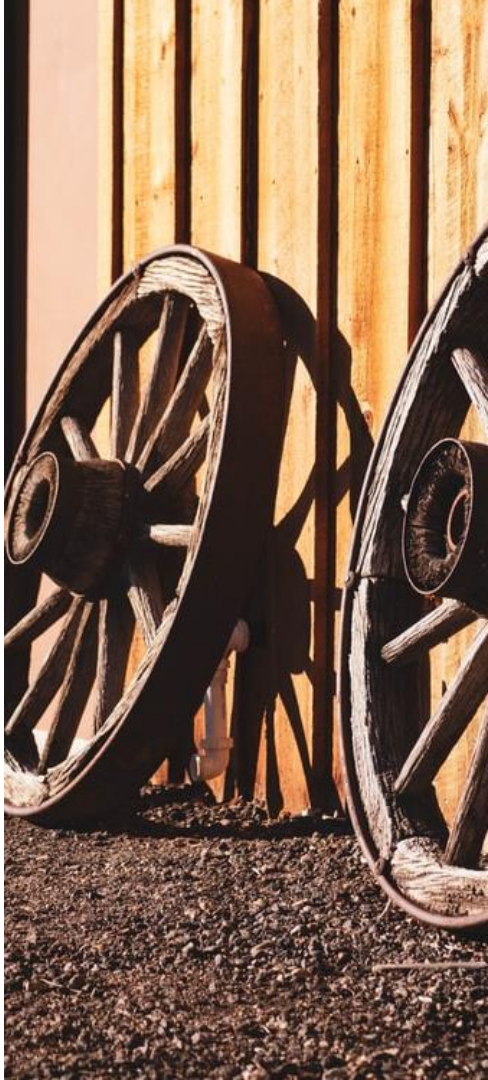
When: Thu, 13:30 – 14:15

Who: Alexander Reelsen (Elastic)

Availability of Secure Frameworks

Do not reinvent the wheel!

- Build on proven technology if possible
- But only pull in what's strictly needed
- Framework availability might influence the language choice



Availability of Secure Frameworks

Typical jobs done by frameworks

- Authentication
- Session management
- Authorization
- Data persistence
- Templating
- Configuration
- …

Availability of Secure Frameworks

Examples of good web frameworks

- **JavaScript**: Express.js, Angular, React, Vue.js
- **Java**: Spring
- **PHP**: Symfony, Laravel
- **Ruby**: Ruby on Rails
- **C#**: .NET Core

Security Criteria for Choosing a Language

- Memory safety
- Type safety
- (Parallelization support)
- Sandbox support
- Availability of secure frameworks

Secure Coding Practices

The basics of secure coding

Secure Coding Practices

- Input handling
- Output handling
- Pitfalls in low-level languages
- The Principle of Complete Mediation
- Cryptography
- Session management
- Concurrency

Input Handling

- **Input handling has three major activities**
  - Canonicalization
  - Input validation
  - Sanitization

Canonicalization

Question: Are someone@example.com and Someone@example.com the same email address?

The answer is a clear **yes and no**.

But again, why should we care?

Canonicalization – Why Should We Care?

- **In the case of email**
  - The uniqueness is often important for security
  - Not canonicalizing it might make impersonation possible
- **Other examples**
  - IP addresses (127.0.0.1 vs. 2130706433)
  - URLs (https://www.a.com vs. https://www.a.com/ vs https://www.a.com:443/ vs. …)

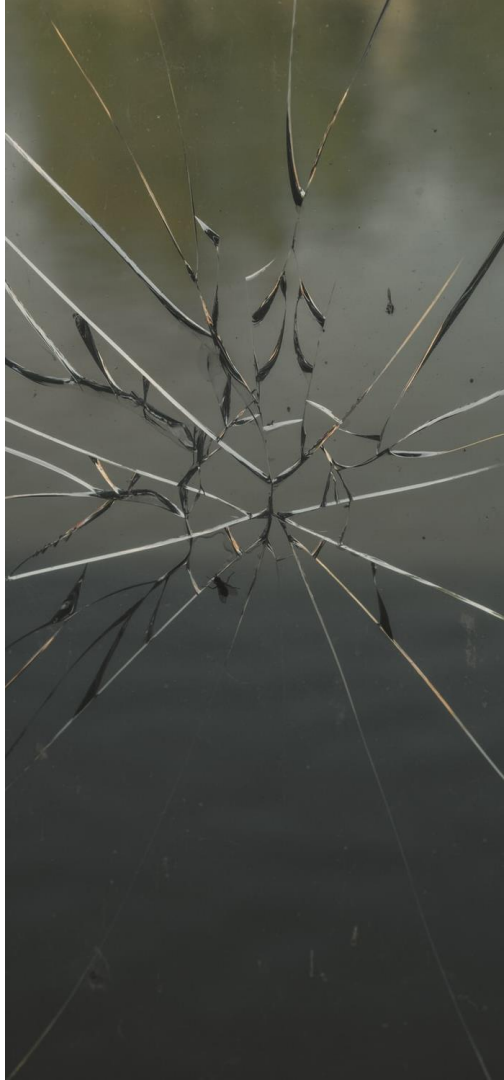
Input Validation

- **Checking inputs against certain formats**
  - Maximum length
  - Allowed characters
  - Date and time
  - Boolean values
  - Email addresses
  - …

Input Validation

Does input validation help against specific vulnerabilities?

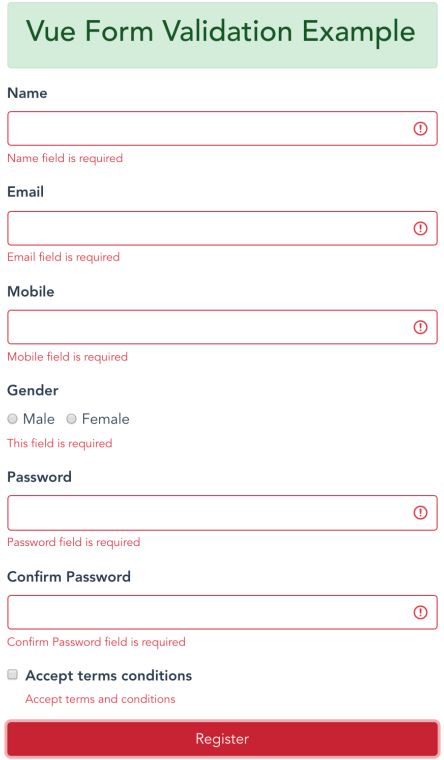
- Usually, no!
- But it’s a good generic measure to reduce the attack surface
- Think SQL injection vs. allowing the ' character
- Output encoding is the key!



Input Validation on the Client Side?

Is there something wrong with client-side validation?

- No, if there is a server-side counterpart
- The web app will be faster as you safe a server round trip per form submission
- Good for usability
- Have both in place!



Output Encoding

- **What is it?**
  - Safely embedding user input into different data structures
  - Converting characters that have a special meaning in the target syntax
  - Avoiding the possibility to change the parent data structure

```
function contactHandler() {
    $('#contactBtn').click(function () {
        var form = $('#__AjaxAntiForgeryForm'
        var token = $('input[name="__RequestV

            "<p>&lt;script&gt;alert('XSS atta

        var message = quill.root.innerHTML;

        message = escapeHtml(message);
        $.ajax({
            url: "/Communication/ContactAdver
            data: { __RequestVerificationToke
            dataType: 'json',
            type: "POST",
        });
    });
}
```

Image source: https://stackoverflow.com/questions/54343557/how-to-display-encoded-html-in-browser

Output Encoding

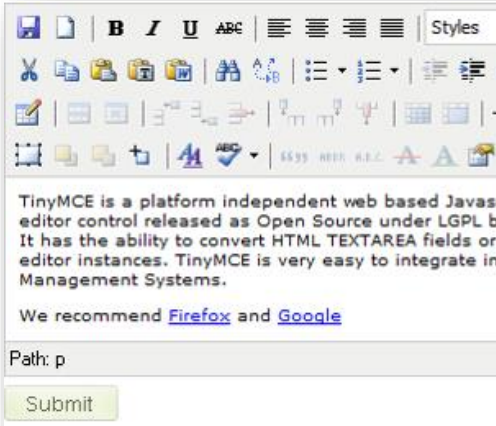
- **What are common situations where I must be aware of the dangers, and encode inputs?**
  - HTML
  - JavaScript
  - XML
  - CSV
  - LDAP
  - SMTP
  - (SQL)
  - …

Output Encoding

- **As opposed to input validation, this is usually the primary protection mechanism!**
  - Input validation lowers the attack surface
  - Output encoding protects against specific vulnerability classes

Sanitization

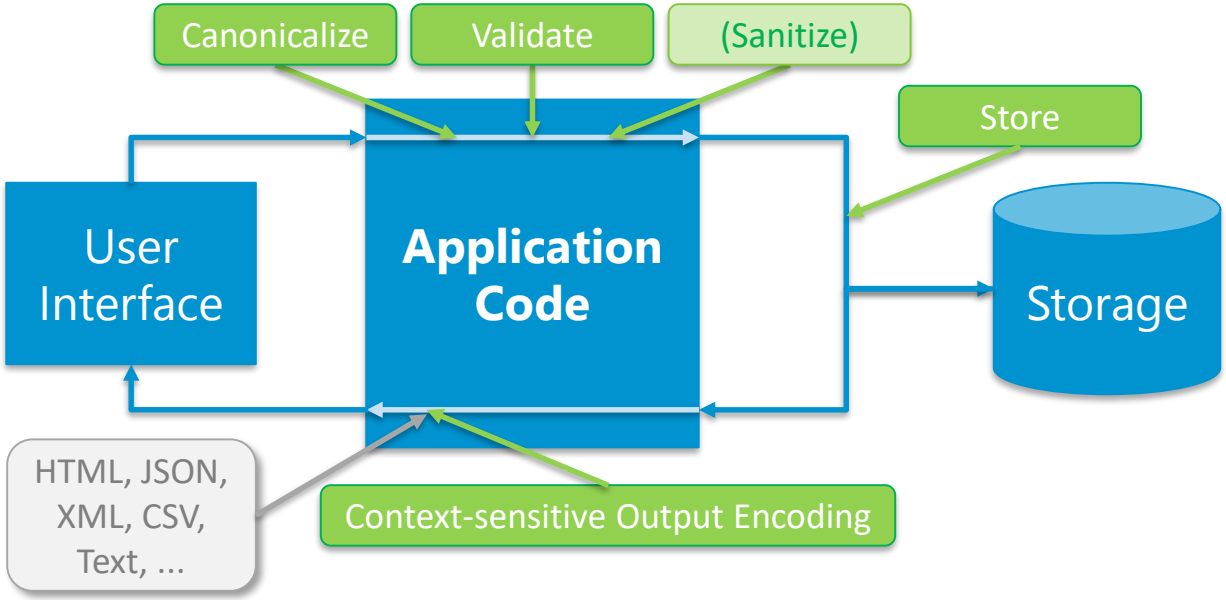
- There will be situations where you **must output HTML directly**
- Think text that can be formatted (WYSIWYG)
- In this case, we must get rid of the "dangerous" parts, e.g., everything that may contain JavaScript
- This is called **sanitization**!



Sanitization: The Rules

- **Don’t roll your own sanitizer!**
  - There are specialized libraries!
  - DOMPurify (JavaScript)
  - Angular comes with its own
  - HTML Purifier (PHP)
  - OWASP Java HTML Sanitizer
  - HtmlSanitizer (C#)
  - …

Canonicalize, Validate, (Sanitize), Store, Encode



Pitfalls in Low-Level Languages

- **Potential vulnerabilities in non-memory-safe languages**
  - Buffer overflows
  - Heap overflows
  - Format string vulnerabilities

Buffer Overflows

- Classical Buffer Overflow

```
void function foo(const char * arg)
{
  char buf[10];
  strcpy(buf, arg);
  ...
}
```

- Buffer Overflow in C++

```
char buf[BUFSIZE];
cin >> (buf);
```

Buffer Overflows: The Problem

- **In RAM: Structured mix of data and code**
  - Program writes beyond memory area
  - Overwriting control structures
  - Modified behavior of the following program flow

Buffer Overflows: Countermeasures

- **What can we do about them?**
  - Don’t write beyond the buffer, do bounds checks!
  - Be careful with user input
  - Use String and Vector classes in C++
  - Do not use unsafe methods like `strcopy`
  - C11/C18 Annex K: Bounds-Checking Interfaces
- **Stay in the “safe world” when using languages like Rust and Go!**

The Principle of Complete Mediation

- **What does that mean?**
  - It means „access control at every request“
  - Always suppose the user knows all API URLs and parameters
  - Be careful with multi-step forms
  - Structure your access control well and centralize it

Cryptography

- **Primitives**
  - Block ciphers (AES, Camellia)
  - Stream ciphers (ChaCha20)
  - Hash functions
  - Public key primitives (Factoring, Elliptic Curves)
- **Schemes**
  - Symmetric crypto systems
  - Asymmetric crypto systems
  - Message authentication code (MAC)
- **Protocols**
  - TLS
  - SSH
  - IPSec
  - S/MIME

Cryptography: End-to-end-Encrypt It All?

End-to-End Encryption: Things to Consider

All these are hard to do

- Key recovery
- Backup
- Multi-device
- Database indexing
- Search
- Scalability
- …

Cryptography: Important Rules

- **Don't roll your own crypto!**
- Don't just check the "encryption" checkbox – be fully aware of the threats and whether crypto can help!
- Use good randomness
- Use AEAD ciphers for integrity in symmetric crypto
- Use unique IVs per cleartext when re-using keys
- Use expensive key derivation when the key base is a human-generated password
- **Good crypto is hard – get help if necessary!**

Session Management

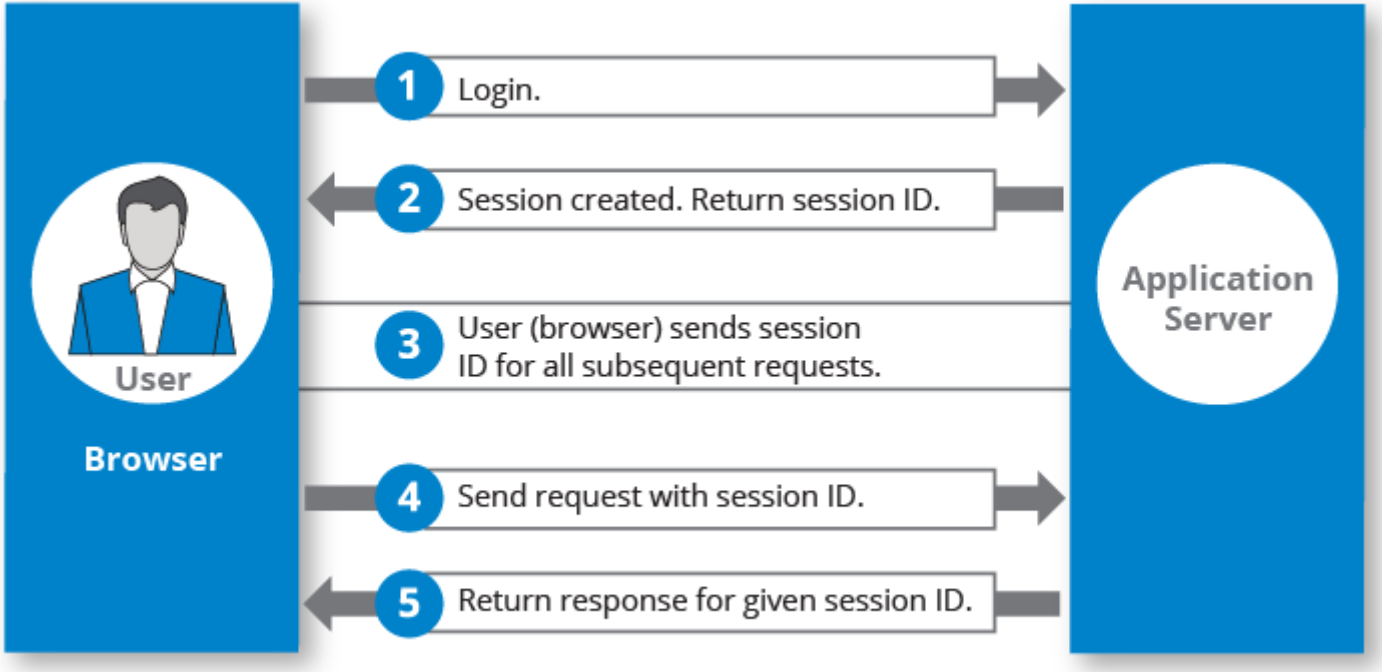


Image source: https://hazelcast.com/glossary/web-session/

Session Management: Important Rules

1. Use your framework's session management if possible
2. Make sure session IDs are non-guessable
3. The session is the only source of information for security decisions
4. Make sure the session ID changes upon successful login
5. Have inactivity and absolute timeouts implemented and configurable

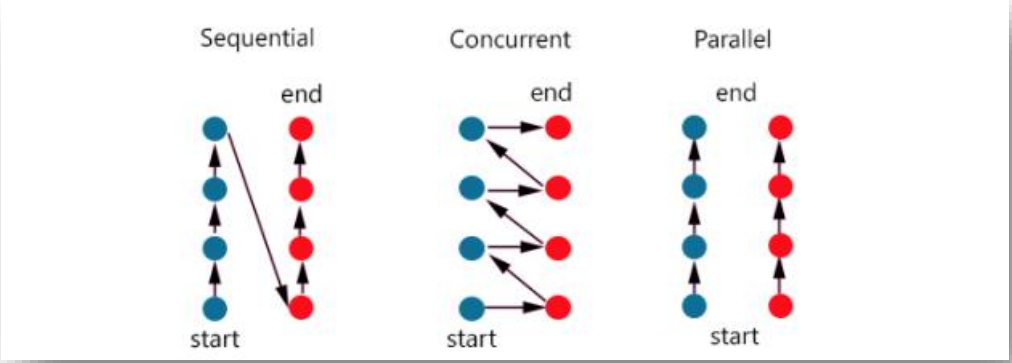
Related sec4dev Talk!

Token Security in Single Page Applications

When: Wed, 10:00 – 10:45

Who: Philippe De Ryck (Pragmatic Web Security, Google Developer Expert)

Sequential, Concurrent, Parallel



https://medium.com/hbot/concurrency-%E0%B8%81%E0%B8%B1%E0%B8%9A-parallelism-%E0%B8%95%E0%B9%88%E0%B8%B2%E0%B8%87%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B8%A2%E0%B8%B1%E0%B8%87%E0%B9%84%E0%B8%87-17dc15ff90f6

Concurrency

Let's discuss these two situations

1. Two users check out the same version of an object in a web application, edit it, and save it.
2. A transaction that may happen simultaneously reads a value from the database, does a computation on it, and writes it back to the database.

Time of Check, Time of Use

Thread 1	Thread 2
($10) function withdraw($amount) { ($10,000) $balance = getBalance(); if($amount <= $balance) { ($9,990) $balance = $balance - $amount; echo "You have withdrawn: $amount";	

	($10)
	function withdraw($amount)
	{ ($10,000)
	$balance = getBalance();
	if($amount <= $balance)
	{ ($9,990)
	$balance = $balance - $amount;
	echo "You have withdrawn: $amount";
	setBalance($balance); ($9,990)
	}
	else
	{
	echo "Insufficient funds.";
	}
	}

setBalance($balance); ($9,990)
}
else
{
echo "Insufficient funds.";
}
}	

Image source: https://defuse.ca/race-conditions-in-web-applications.htm

Race Condition

https://society6.com/product/dont-be-happy-worry_t-shirt

Concurrency: Solution Approaches

Ask yourself: Will such situations be the exception or the rule?

- Exception: Optimistic measures
- Rule: Pessimistic measures

Concurrency: Solution Approaches

Possible Solutions

- Entity versioning, exception on mismatch (optimistic)
- Atomic operations (avoid race windows)
- Mutual exclusions
  - File locks
  - DB (row-level) locks
- Message passing

Secure Coding Practices

- Input handling
- Output handling
- Pitfalls in low-level languages
- The Principle of Complete Mediation
- Cryptography
- Session management
- Concurrency

Clean Code

Readability, maintainability, testability, and how they relate to security

Why Even Bother?

What is the most important prerequisite for you as a tester to assess the security of a piece of software?

Make your guess in the session chat!

It's solid understanding of both its functionality and its context.

Why Even Bother?

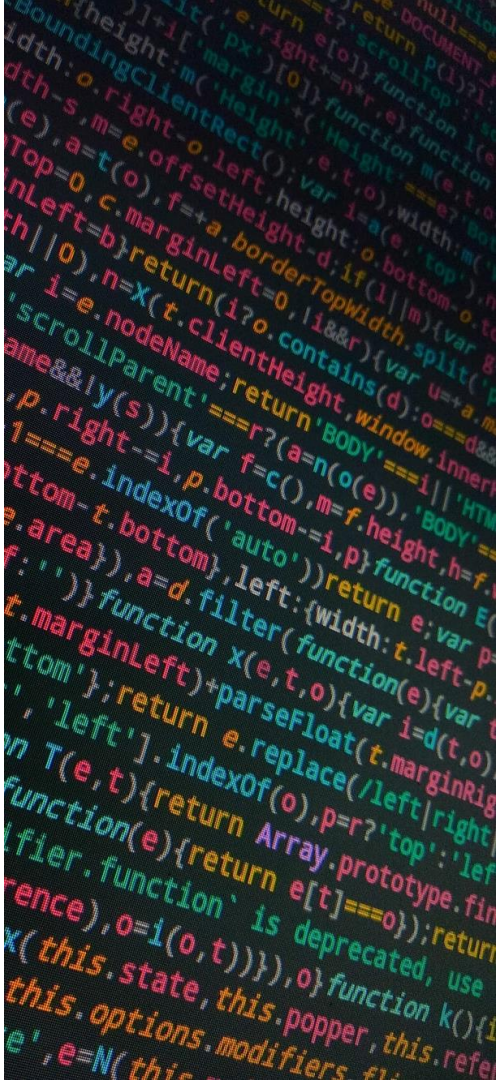
- **Unreadable code tends to be insecure**
  - Unreadable code is hard to understand
  - No understanding means creative thinking about how to circumvent security measures is basically impossible
- **Unreadable code adds to your technical debt**

SBA Research, 2020

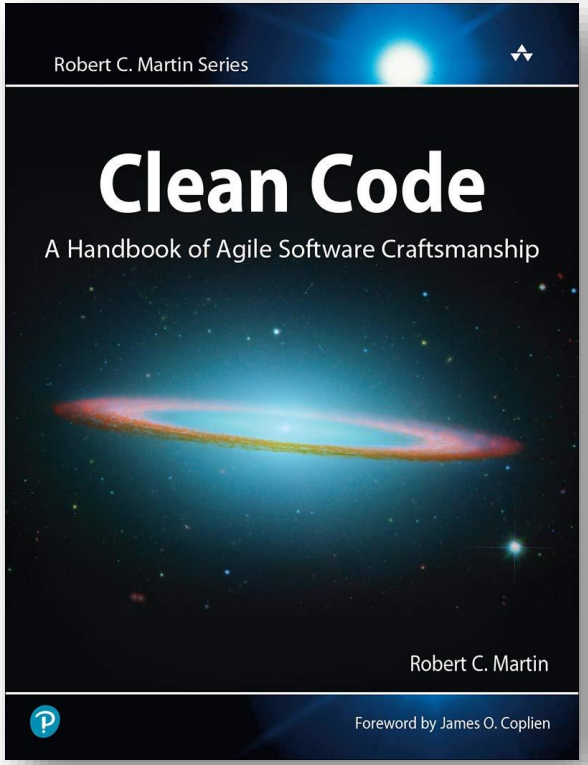
Principles of Readable Code

1. Single responsibility
2. Well-structured
3. Thoughtful naming
4. Simple and concise
5. Comments explain „why“, not „how“
6. Continuously refactored for readability
7. Well-tested

Source: https://blog.pragmaticengineer.com/readable-code/



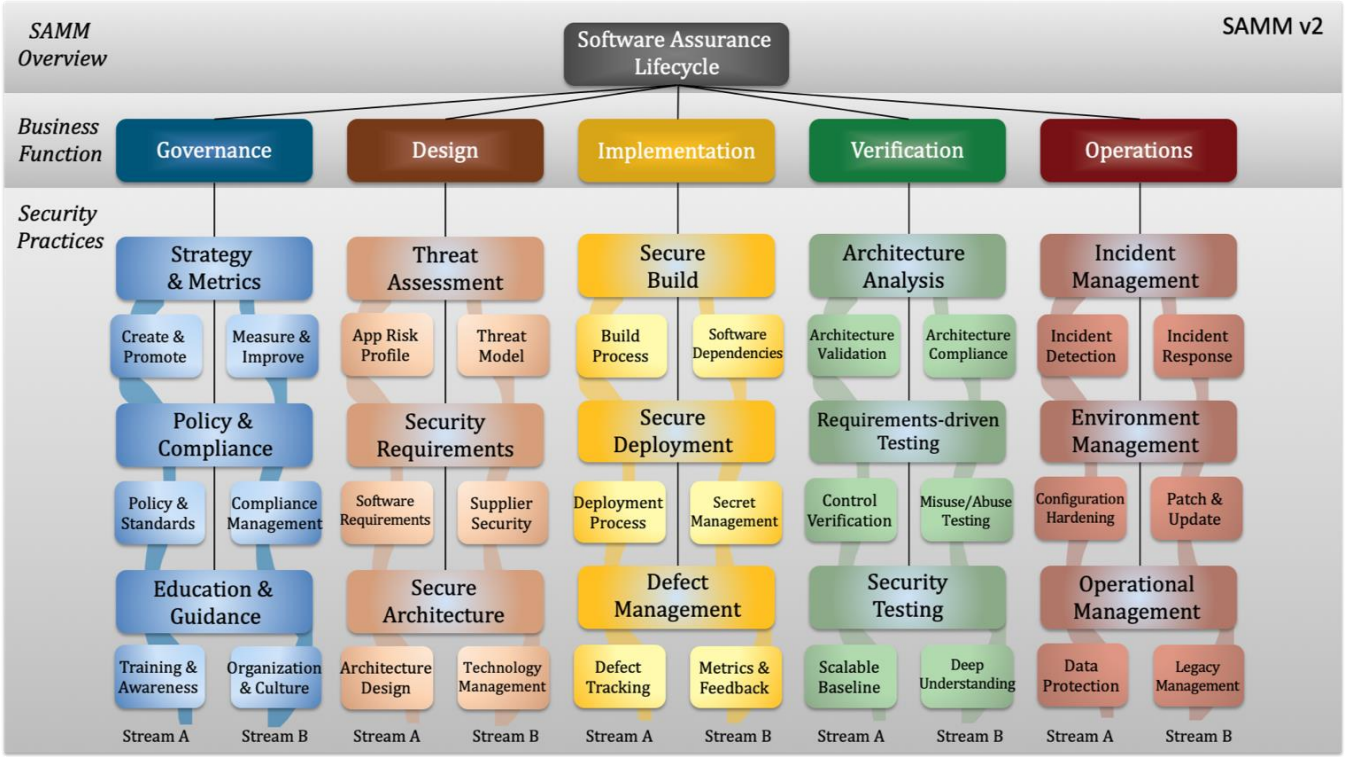
Book Recommendation: Clean Code



Secure Software Development Lifecycle (SDLC) Fundamentals

OWASP SAMM, shifting left, examples

Secure SDLC Fundamentals: OWASP SAMM



Free Talk on SAMM at SBA Live Academy

Related sec4dev Talk!

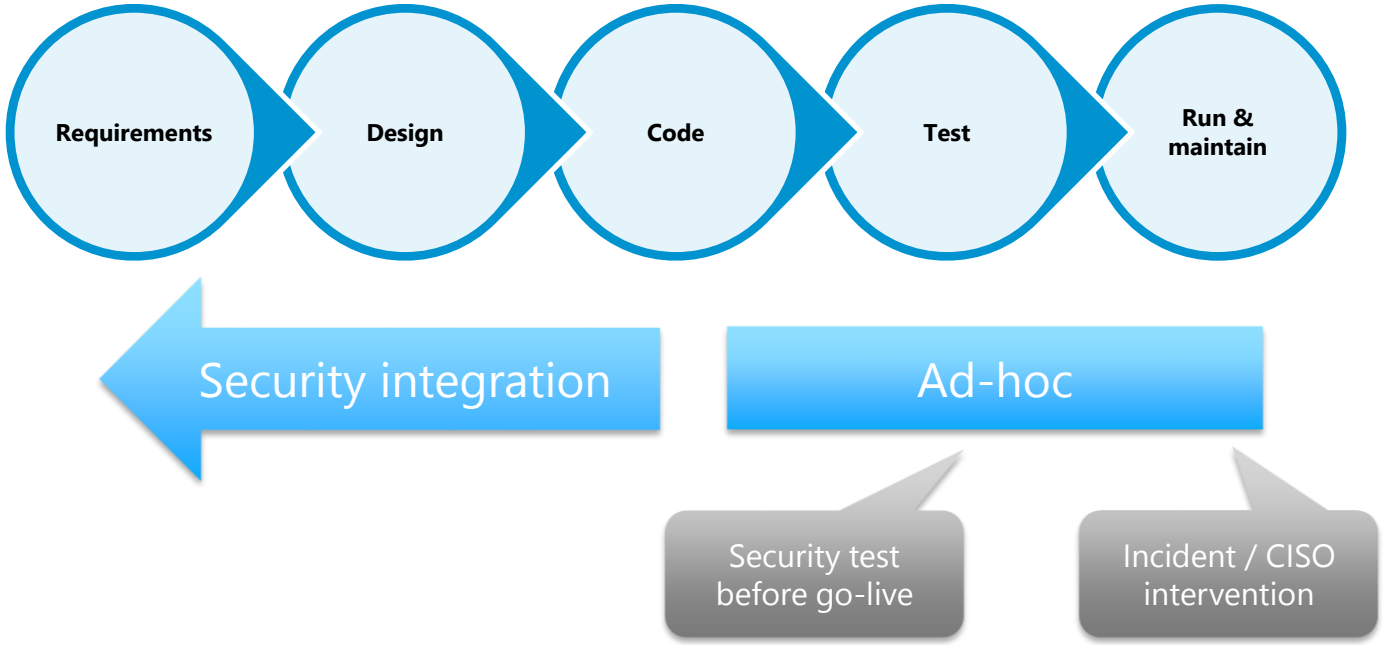
Keynote: Security Metrics That Matter

When: Wed, 17:15 – 18:00

Who: Tanya Janca
(We Hack Purple, OWASP)

Shifting Left

Systematic approach



Related sec4dev Talk!

So Happy Together: Making the Promise of DevSecOps a Reality

When: Thu, 17:15 – 18:00

Who: Alyssa Miller
(S&P Global Ratings)

Education & Guidance

Expert

- Certified Information Systems Security Professional (CISSP)
- Certified Secure Software Lifecycle Professional (CSSLP)

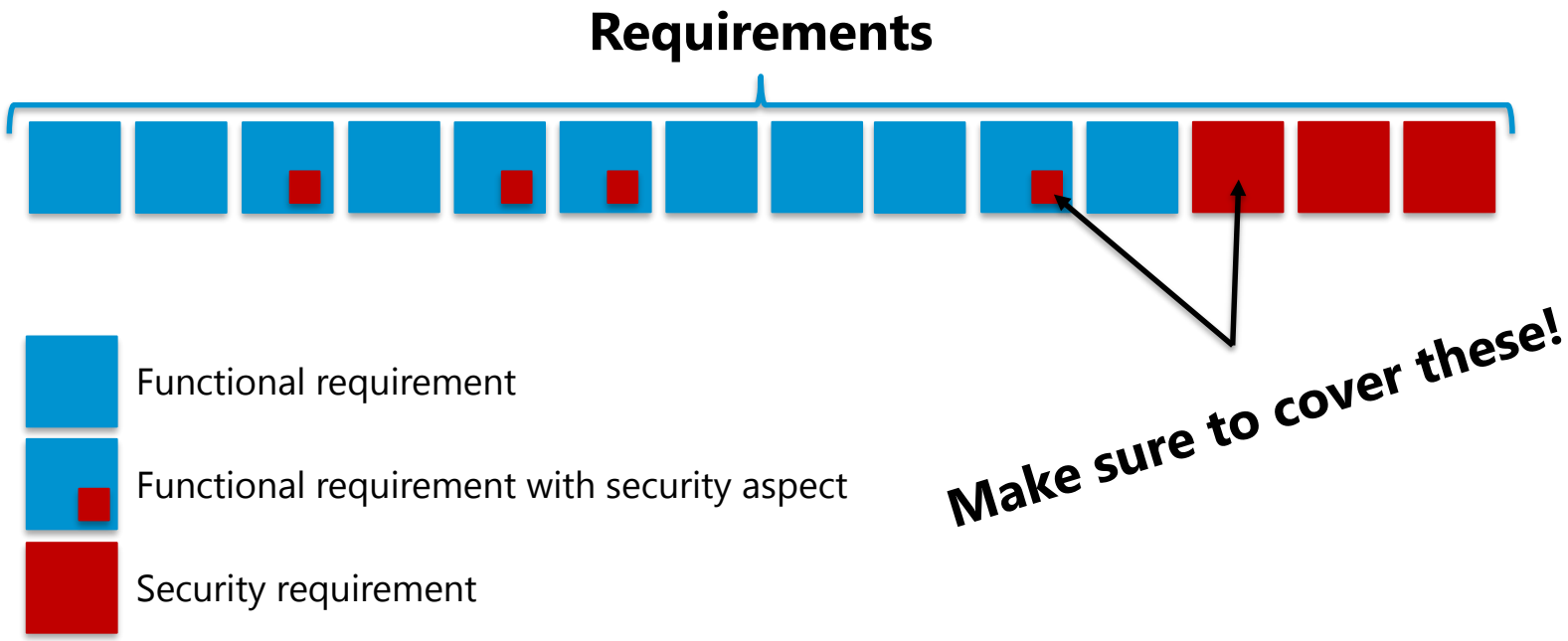
Advanced
Pick your area

- C / C++ Security
- Threat Modeling
- Secure Coding
- Cloud Security
- Web App Security
- IoT Security

Basic

- Secure SDLC Essentials

Requirements-Driven Testing



Threat Assessment

Example: Typical account security threat model

Threat	Severity[1]	C/I/A	Countermeasures
Password guessing	**High**	C/I/-	(Temporary) user lockout, password policy, MFA, transparency (device lists and notifications, with Device Tokens)
Account lockout	**Medium**	-/-/A	Selective lockout (with Device Tokens)
Misuse of known passwords (public lists, other apps, ...)	**Medium**	C/I/-	Multi-factor authentication
Someone dumps the DB on the Internet	**Medium**	C/I/-	Proper hashes (Argon2)
Enumerating valid usernames	**Low**	C/-/-	(Generic error messages, constant timing on all requests containing the username)

[1] *The severity really depends on the classification of your data. Don't see them as absolute and unchangeable values.*

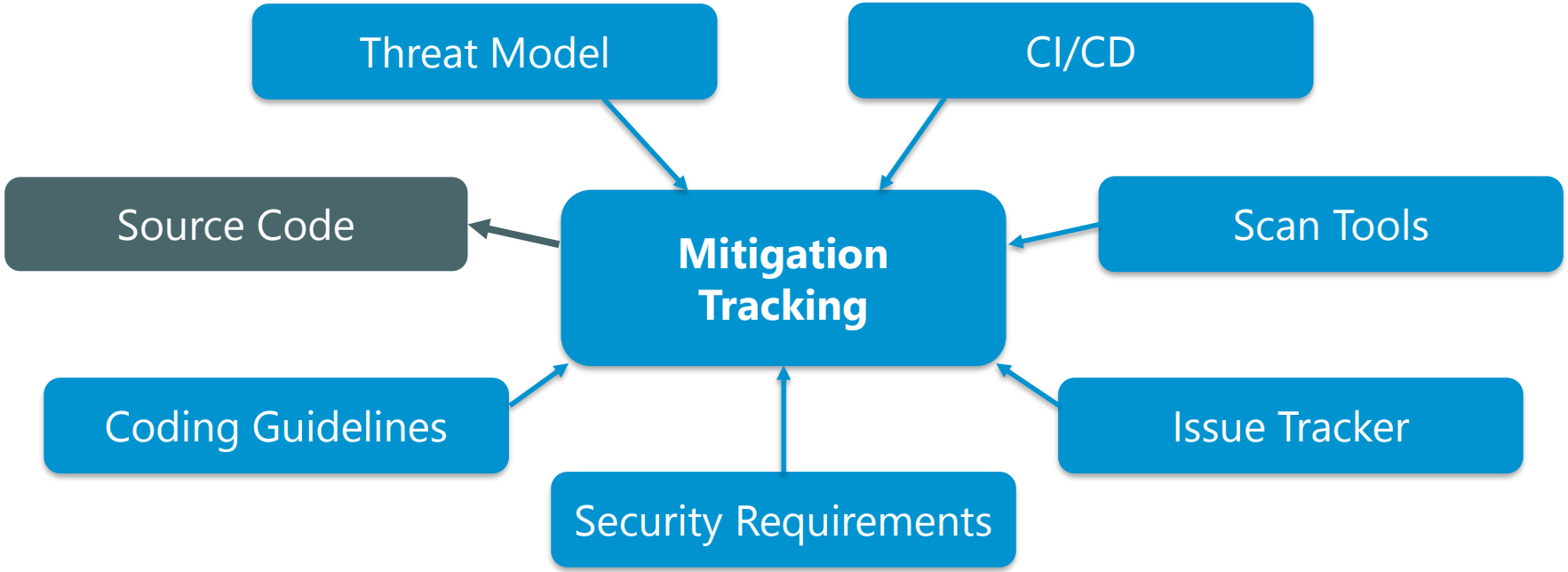
Related sec4dev Talk!

Rapid Risk Assessment: A Lightweight Approach

When: Wed, 14:15 – 15:00

Who: Julien Vehent (Cloud Security, Google's Detection and Response team, formerly Mozilla)

Defect Management



Automated Tool Types

- **Static Application Security Testing** (SAST)
- **Dynamic Application Security Testing** (DAST)
- **Interactive Application Security Testing** (IAST)
- **Dependency Checks** (DC, no, just kidding)

Static Application Security Testing (SAST)

- Scans the source code
- No running application required
- Builds a so-called Abstract Syntax Tree (AST)
- **Approach**: Input – way through your code – sink

SAST: Abstract Syntax Tree

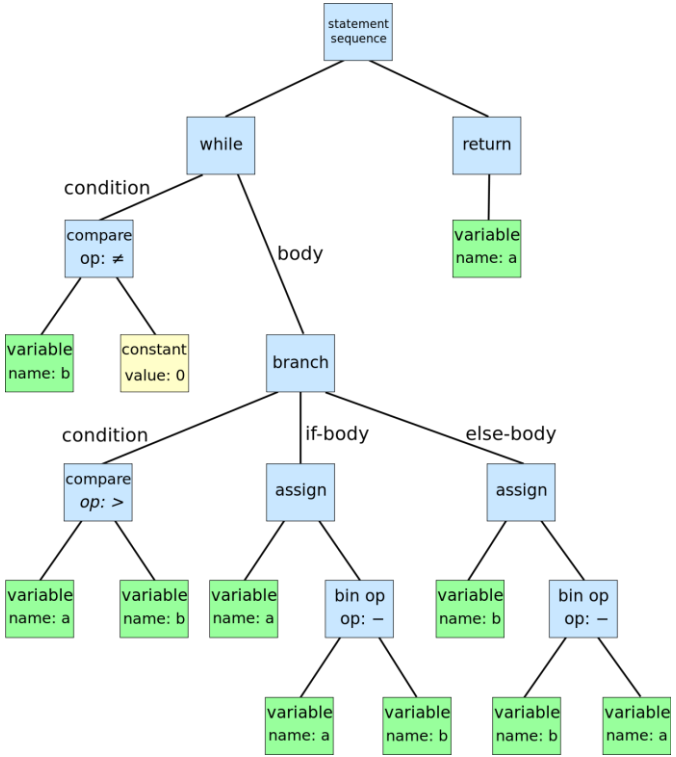


Image source: https://en.wikipedia.org/wiki/Abstract_syntax_tree

Static Application Security Testing (SAST)

- **Advantages**
  - Reproducible results
  - Good code coverage
- **Disadvantages**
  - Usually only covers your own code
  - Can only detect a limited set of vulnerabilities
  - Lacks context when scanning microservices

Microservices and Vulnerability Context

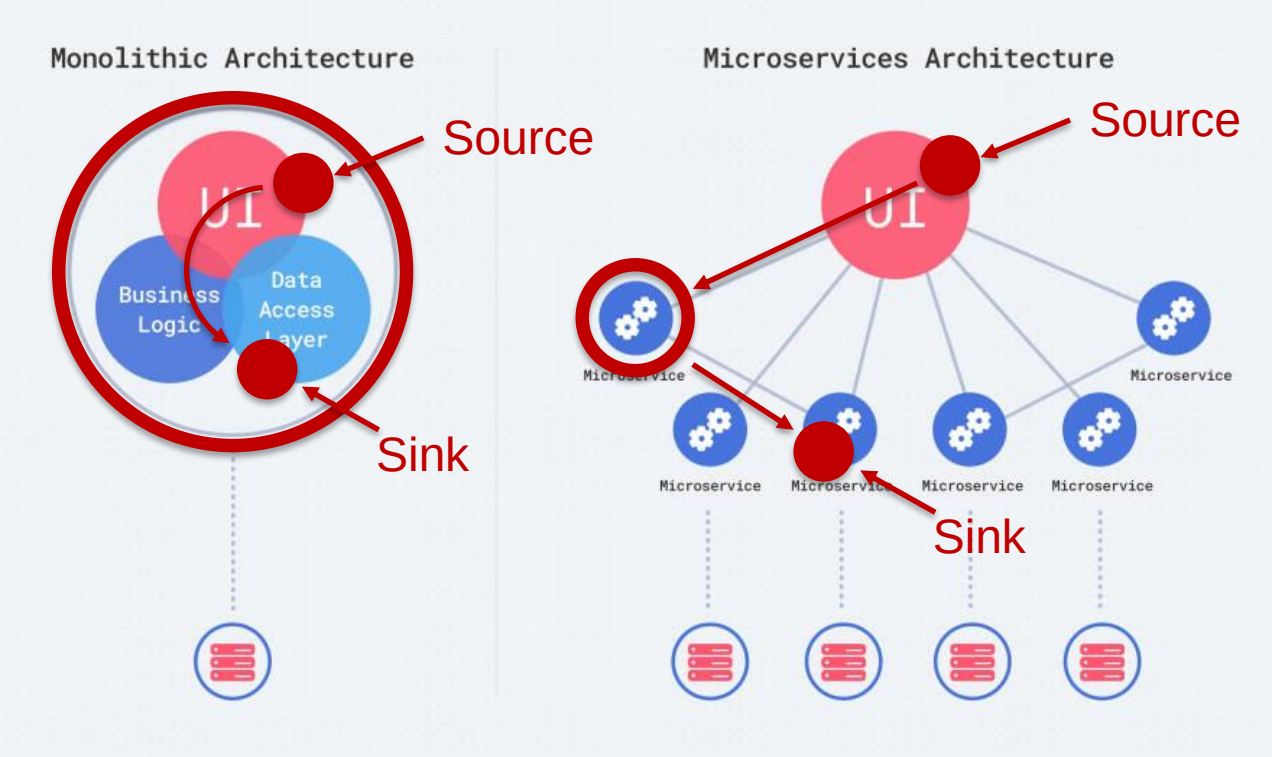


Image source: https://dev.to/alex_barashkov/microservices-vs-monolith-architecture-4l1m

Related sec4dev Talk!

Know Your Tools: Quirks And Flaws Of Integrating SAST Into Your Pipeline

When: Wed, 10:45 – 11:30

Who: Artem Bychkov (Advanced Software Technology Lab, Huawei)

Dynamic Application Security Testing (DAST)

- Scans a running application
- Some tools have SAST elements built in
- **Approach**: Request – response

Interactive Application Security Testing (IAST)

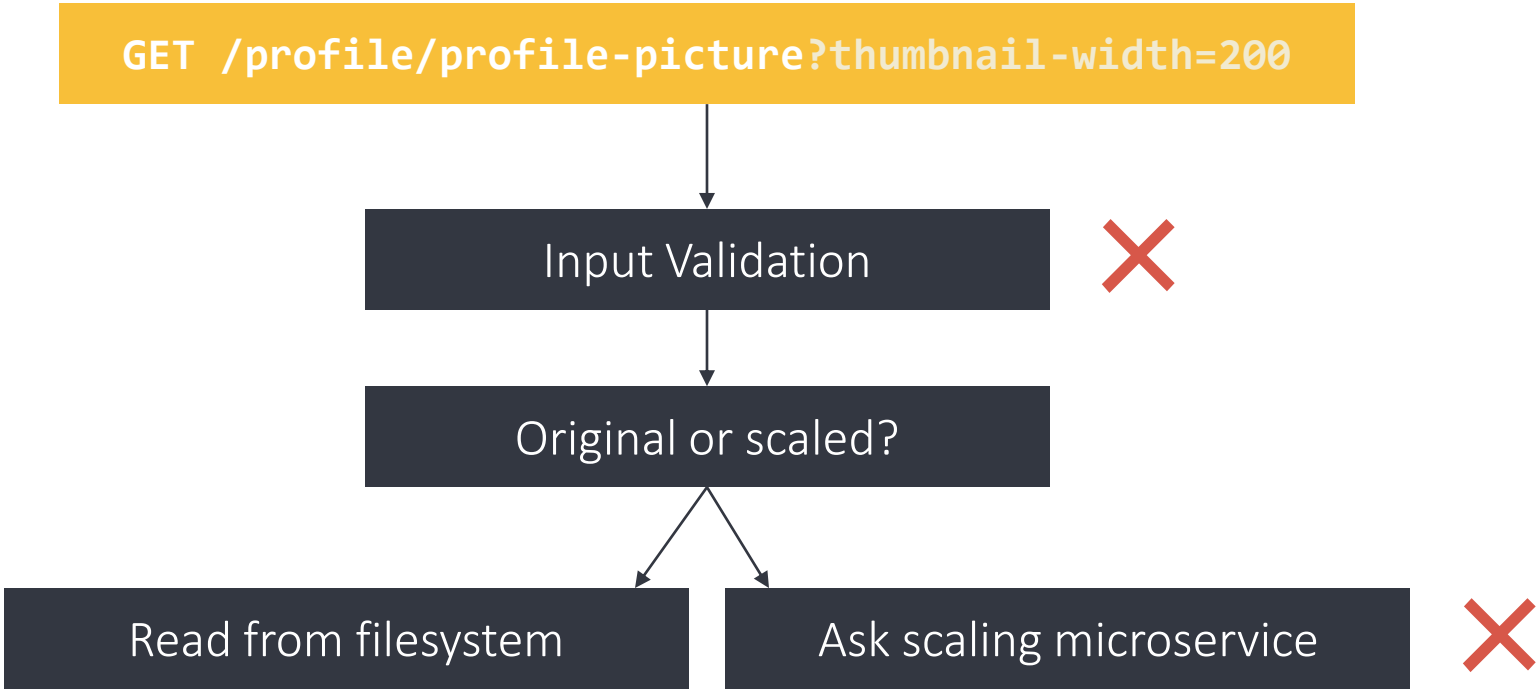
- During a dynamic scan (DAST), an agent is instrumented into the application runtime
- Agent has insight into the logic flow
- Makes DAST results more actionable
- Runtime Application Self-Protection (RASP)

 Image: https://www.e-spincorp.com/documentation/iast-interactive-application-security-testing/

Dynamic Application Security Testing (DAST)

- **Advantages**
  - Touches more parts of your stack
  - Tends to have less false positives
- **Disadvantages**
  - Can only detect a limited set of vulnerabilities
  - SPAs require heavy lifting (headless browser)
  - Hard to get good code coverage

Dynamic Tests: Known-Good Requests



SAST vs. DAST

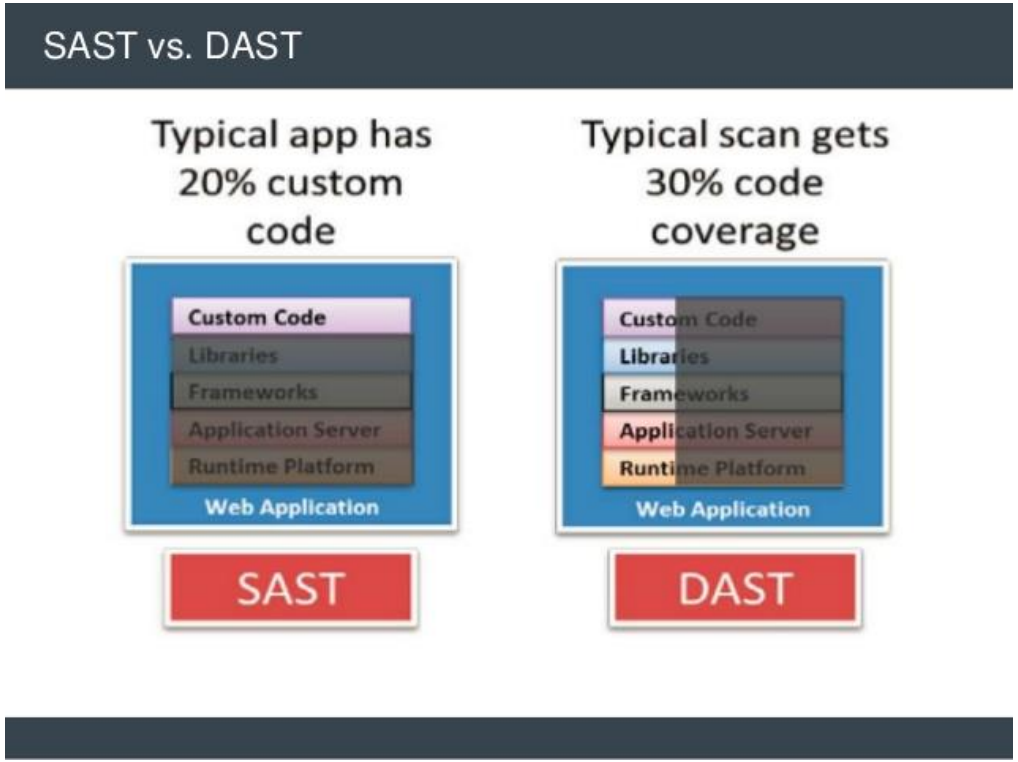


Image source: https://www.slideshare.net/cisoplatform7/application-security-architecture-and-threat-modelling

General Tool Weaknesses

- Tools might know that “injection is bad” but not that “this user must not see this dataset”
- How would a tool know what functionality a role can call?
- Design flaws cannot be detected

Different Tools Give Different Results

- Make sure to use a variety of views on your software
- A penetration test is usually a good start
- Automate bit by bit, don’t mindlessly throw expensive tools at your software

OWASP SAMM

Output and results

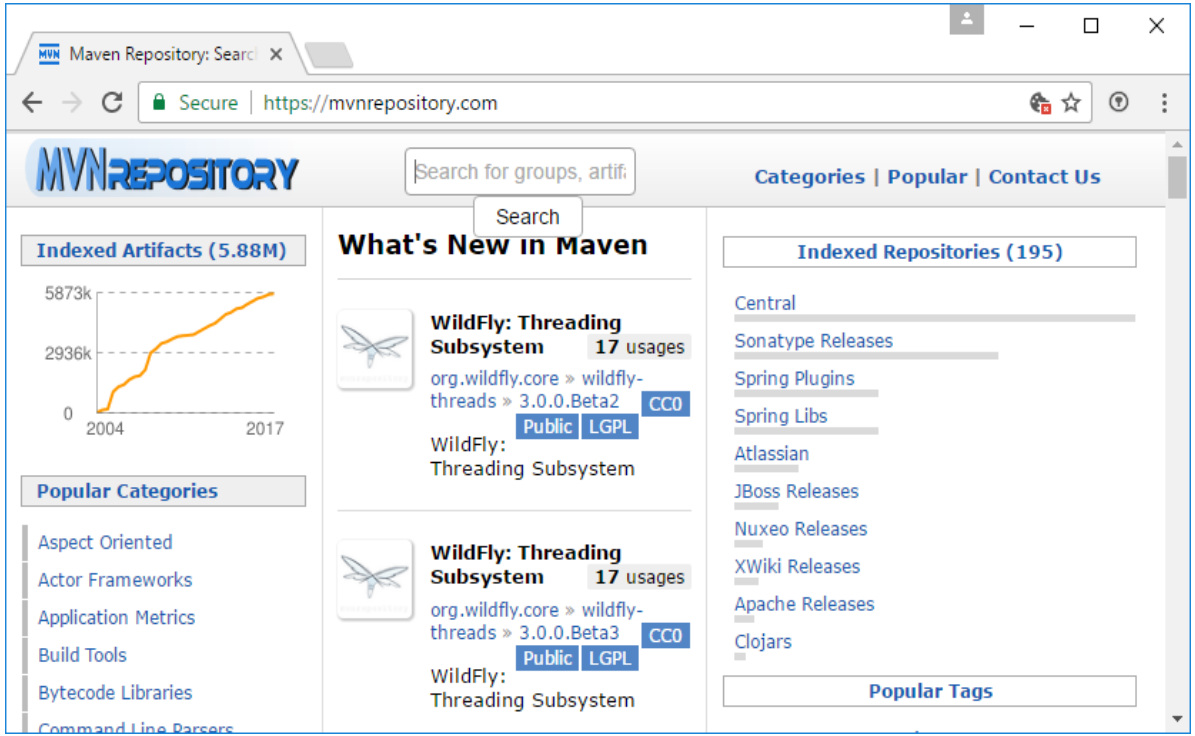
- **What you get**
  - A scored result for each function
  - Every activity has the same weight
  - Every level has the same weight
- **Score is not the ultimate goal**
  - Rather the road map and process resulting from it
  - Detect blind spots

			Current Maturity Score		
			Maturity		
Functions	Security Practices	Current	1	2	3
Governance	Strategy & Metrics	0,63	0,25	0,25	0,13
Governance	Policy & Compliance	0,63	0,50	0,13	0,00
Governance	Education & Guidance	0,75	0,38	0,13	0,25
Design	Threat Assessment	0,50	0,25	0,25	0,00
Design	Security Requirements	0,25	0,25	0,00	0,00
Design	Secure Architecture	0,88	0,50	0,13	0,25
Implementation	Secure Build	1,88	1,00	0,63	0,25
Implementation	Secure Deployment	1,13	0,75	0,38	0,00
Implementation	Defect Management	0,63	0,63	0,00	0,00
Verification	Architecture Assessment	0,88	0,75	0,00	0,13
Verification	Requirements Testing	0,75	0,25	0,25	0,25
Verification	Security Testing	1,50	0,75	0,50	0,25
Operations	Incident Management	0,13	0,13	0,00	0,00
Operations	Environment Management	0,50	0,38	0,13	0,00
Operations	Operational Management	1,25	1,00	0,13	0,13

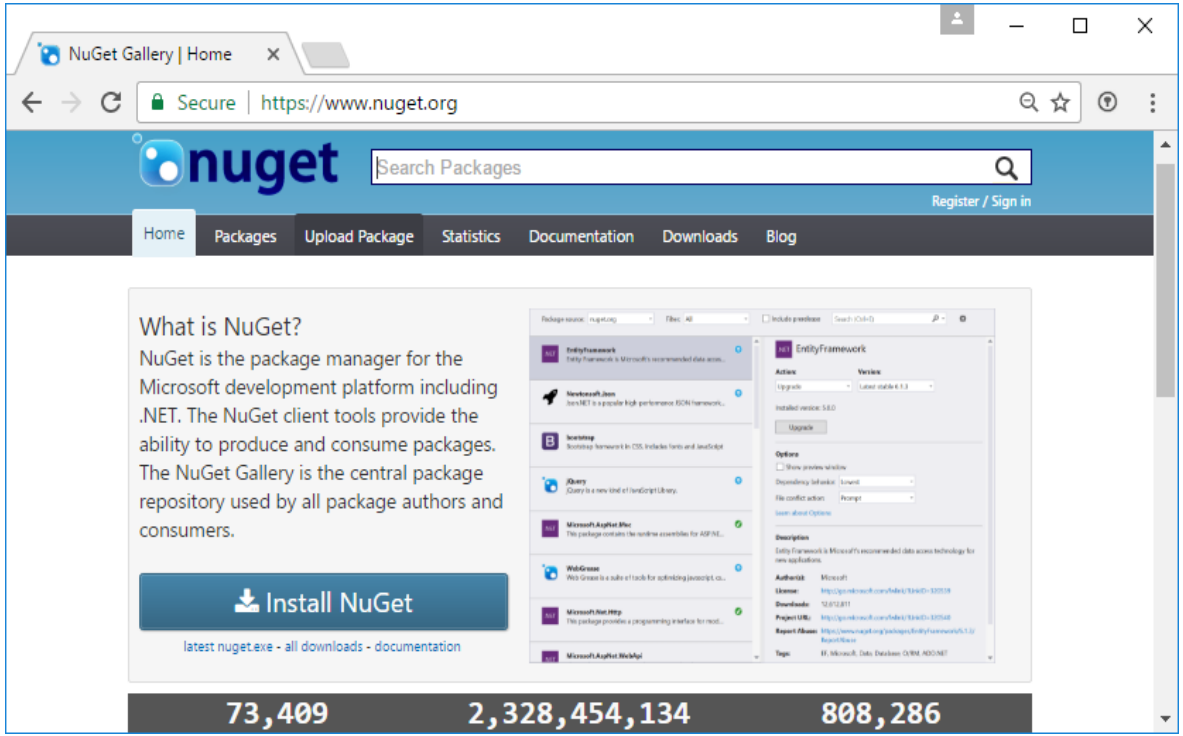
Dependency Management

How to deal with external code

What is a software component?



What is a software component?



What is a software component?

PyPI - the Python Packag

Python Software Foundation [US] | https://pypi.python.org/pypi

python™

search

» Package Index

PACKAGE INDEX

Browse packages
Package submission
List trove classifiers
RSS (latest 40 updates)
RSS (newest 40 packages)
PyPI Tutorial
PyPI Security
PyPI Support
PyPI Bug Reports
PyPI Discussion
PyPI Developer Info

ABOUT

NEWS

PyPI - the Python Package Index

The Python Package Index is a repository of software for the Python programming language. There are currently **99170** packages here.
To contact the PyPI admins, please use the Support or Bug reports links.

Not Logged In

Login
Register
Lost Login?
Use OpenID
Login with Google

Status

Nothing to report

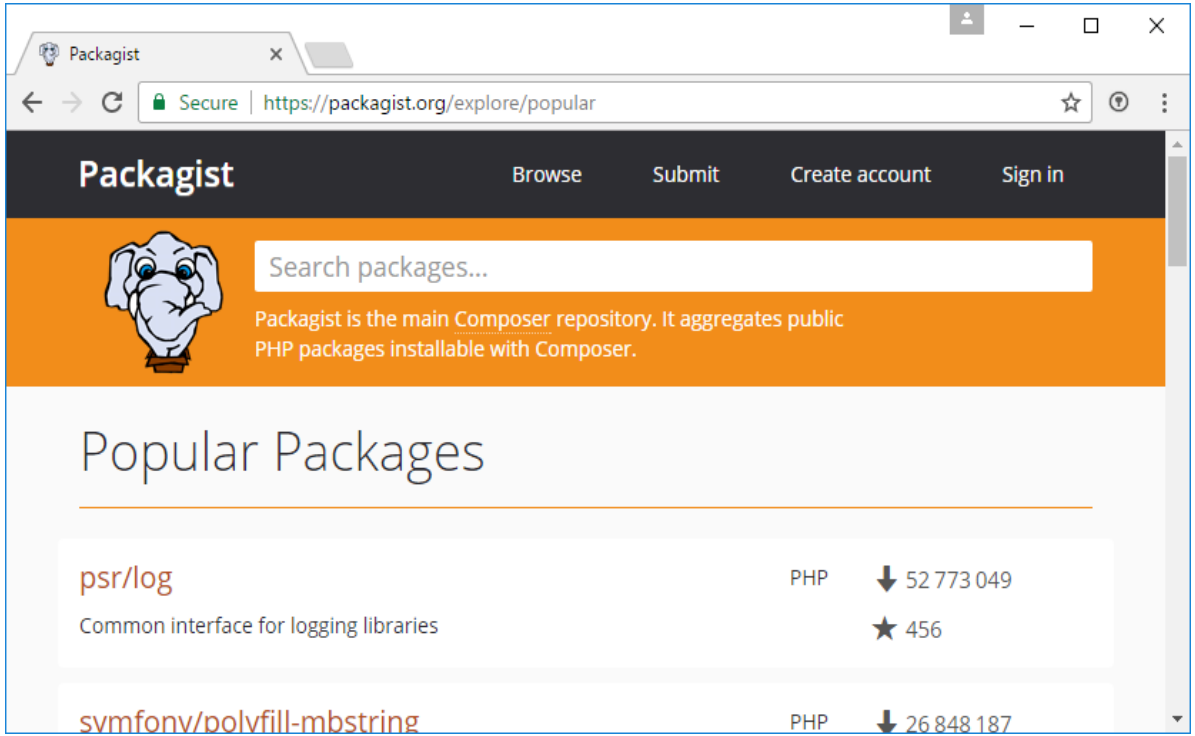
Get Packages

To use a package from this index either "pip

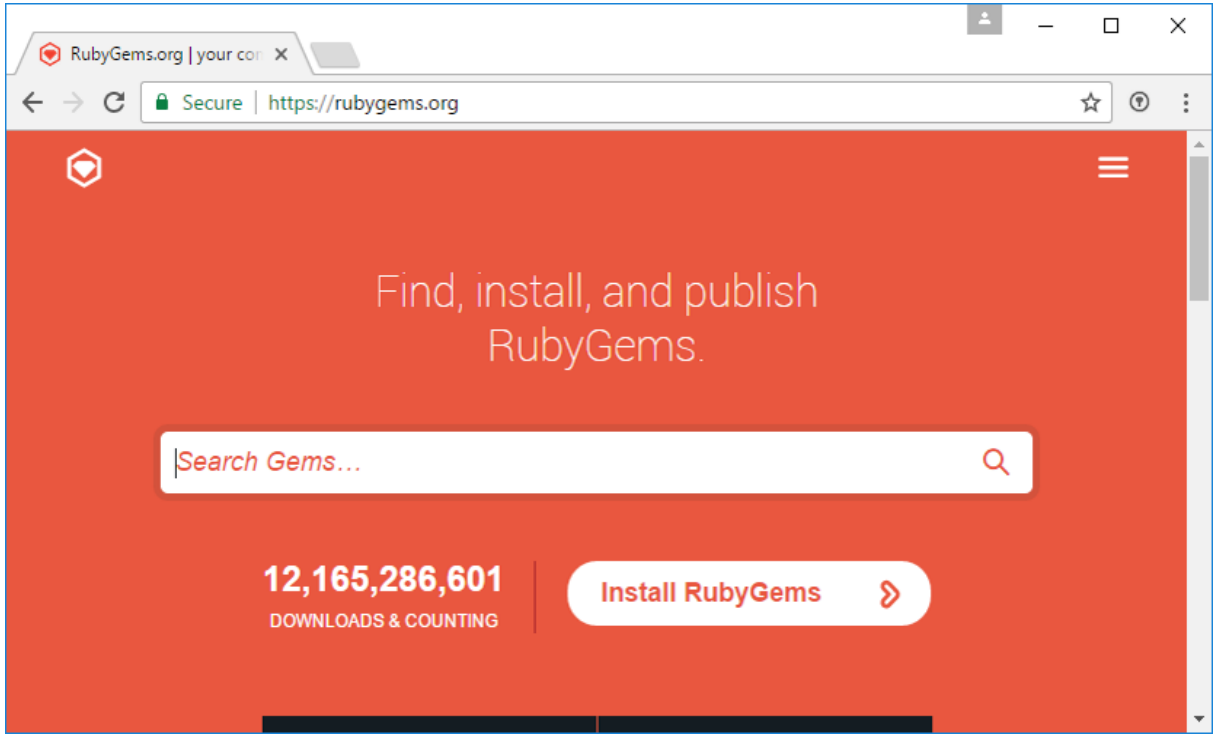
Package Authors

Submit packages with "python setup.py

What is a software component?



What is a software component?



Foreign Code Usually Prevails

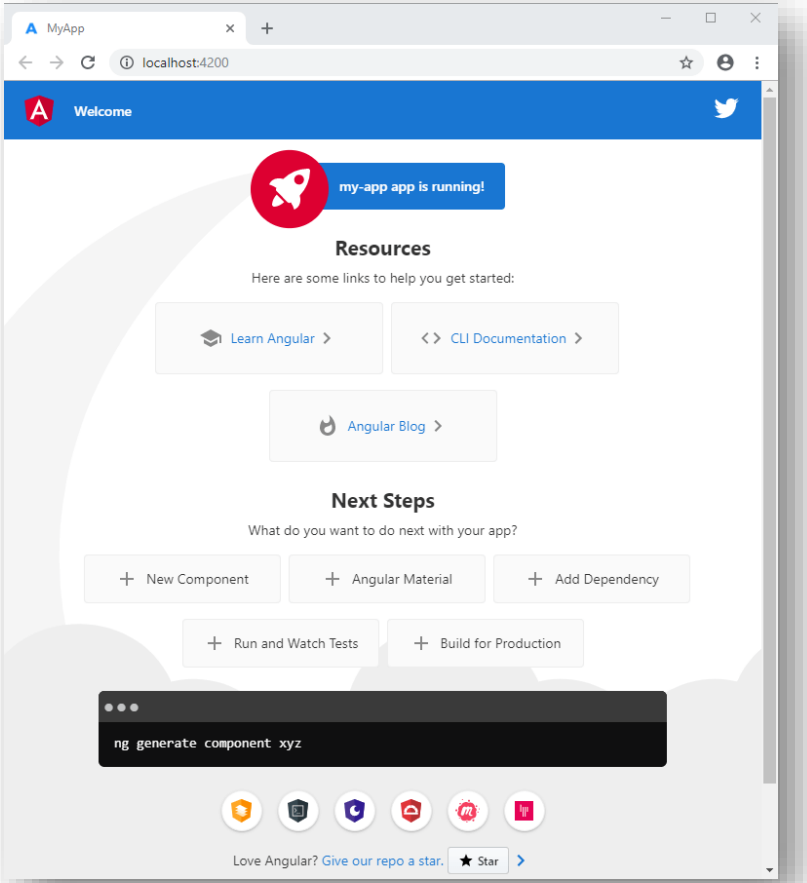
Foreign code usually makes up for **> 50 % of running code!**

- We cannot check every line of code
- But we can check them for known vulnerabilities
- Dependencies must be declared machine-readable!

Angular Project with Router and SCSS

```
> cloc node_modules
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JavaScript                   16214         171274         786507        3076493
JSON                          1887            298              0         247588
Markdown                      1628          73253              4         177074
TypeScript                    3069          16591         128264         153548
HTML                           227          13191            214          25464
CSS                            135            380           2275          22039
```

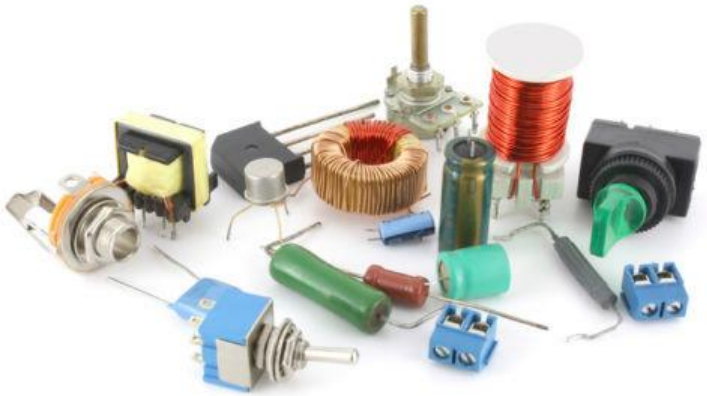
That's 3.2 M LoC in node_modules



https://angular.io/guide/setup-local

Components With Vulnerabilities

- **Components in software**
  - Libraries
  - Frameworks
  - Runtimes (JVM)
  - Base images (Docker)



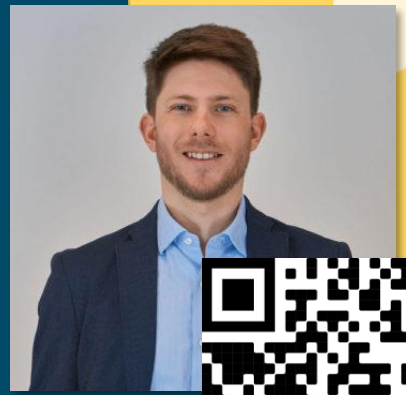
- **Vulnerabilities**
  - Quality of components varies
  - Security Awareness does not always exist
  - Many packages become orphaned/unmaintained
  - Recursive dependencies increase the problem

Related sec4dev Talk!

Let's Build And Break A Container By Hand Without Docker Or LXC

When: Thu, 14:15 – 15:00

Who: Reinhard Kugler (SBA Research)



Automate Dependency Checks!

1. Trigger them **automatically** on every `git push`
2. **Fail** the build!
3. Do it **regularly** even if no pushes happen!

Dependency Checks: Tools

- OWASP Dependency Check (open source)
- npm audit (NPM)
- RetireJS (JavaScript)
- Local PHP Security Checker (PHP, Composer)
- NuGetDefense (.NET)
- dotnet-retire (.NET)
- Safety (Python)
- GitLab (through Gemnasium integration)
- GitHub (through Dependabot)
- Hakiri (Ruby; commercial)
- Snyk Open Source Security Management (commercial)
- JFrog Xray (commercial)
- Sonatype Nexus (commercial)
- Synopsis Black Duck (commercial)

Developer's Checklist: Dependencies

- ❑ Choose your dependencies wisely
- ❑ Have a declarative, machine-readable list of dependencies
- ❑ Check your dependencies in an automated way
- ❑ Fail the build if there are severe vulnerabilities
- ❑ Rebuild and run checks regularly even if there is no push
- ❑ **Advanced**
  - ❑ Have a good test coverage
  - ❑ A bot submits a pull request with updates
  - ❑ Merge it automatically if tests are green

Common Vulnerability Classes

Most common vulnerabilities and their automated testability

Web Applications: OWASP Top 10

OWASP Top 10 - 2013	→	OWASP Top 10 - 2017
A1 – Injection	→	A1:2017-Injection
A2 – Broken Authentication and Session Management	→	A2:2017-Broken Authentication
A3 – Cross-Site Scripting (XSS)	↘	A3:2017-Sensitive Data Exposure
A4 – Insecure Direct Object References [Merged+A7]	∪	A4:2017-XML External Entities (XXE) [NEW]
A5 – Security Misconfiguration	↘	A5:2017-Broken Access Control [Merged]
A6 – Sensitive Data Exposure	↗	A6:2017-Security Misconfiguration
A7 – Missing Function Level Access Contr [Merged+A4]	∪	A7:2017-Cross-Site Scripting (XSS)
A8 – Cross-Site Request Forgery (CSRF)	☒	A8:2017-Insecure Deserialization [NEW, Community]
A9 – Using Components with Known Vulnerabilities	→	A9:2017-Using Components with Known Vulnerabilities
A10 – Unvalidated Redirects and Forwards	☒	A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

Web APIs: OWASP API Security Top 10

API1:2019 Broken Object Level Authorization

API2:2019 Broken User Authentication

API3:2019 Excessive Data Exposure

API4:2019 Lack of Resources & Rate Limiting

API5:2019 Broken Function Level Authorization

API6:2019 Mass Assignment

API7:2019 Security Misconfiguration

API8:2019 Injection

API9:2019 Improper Assets Management

API10:2019 Insufficient Logging & Monitoring

SANS CWE Top 25

Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38
[16]	CWE-732	Incorrect Permission Assignment for Critical Resource	6.95
[17]	CWE-94	Improper Control of Generation of Code ('Code Injection')	6.53
[18]	CWE-522	Insufficiently Protected Credentials	5.49
[19]	CWE-611	Improper Restriction of XML External Entity Reference	5.33
[20]	CWE-798	Use of Hard-coded Credentials	5.19
[21]	CWE-502	Deserialization of Untrusted Data	4.93
[22]	CWE-269	Improper Privilege Management	4.87
[23]	CWE-400	Uncontrolled Resource Consumption	4.14
[24]	CWE-306	Missing Authentication for Critical Function	3.85
[25]	CWE-862	Missing Authorization	3.77

https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

Automated Testability (Very Roughly)

1. WEB1/API8 Injection
2. WEB2/API2 Broken Authentication
3. WEB3 Sensitive Data Exposure
    1. Protection in Transit: TLS
    2. Protection at Rest: API3 Excessive Data Exposure
4. API4 Lack of Resources & Rate Limiting
5. WEB4 XML External Entities (XXE)
6. WEB5 Broken Access Control
    1. API1: Broken Object-Level Authorization
    2. API5: Broken Function-Level Authorization
7. API6 Mass Assignment
8. WEB6/API7 Security Misconfiguration
9. WEB7 Cross-Site Scripting (XSS)
10. WEB8 Insecure Deserialization
11. WEB9 Using Components with Known Vulnerabilities
12. API9 Improper Assets Management
13. WEB10/API10 Insufficient Logging & Monitoring
14. Cross-Site Request Forgery (CSRF)

Vendor Claims: Have A Close Look

- **“We cover the OWASP Top 10”**
  - Well… probably not.
  - Maybe specific aspects of each
  - But even that is highly optimistic
- **Be aware of the false sense of security!**

Steps Towards More Targeted Protection

1. Determine the impact of security incidents (BIA).
2. Document your tech stack.
3. Determine relevant vulnerability types for each component.
4. Find out
    1. how technology helps,
    2. how it could still go wrong,
    3. which tests would catch which errors,
    4. whether defense in depth is implemented,
    5. and what residual risk is left.
5. Document this as close to your daily work as possible.

Example 1: Angular and XSS

- SPAs introduce good separation of concerns
- Only a small set of vulnerabilities is relevant
- **In short: It's mostly XSS**
- Ways to screw up
  - `bypassSecurityTrust*`
  - Direct access of unsafe DOM APIs

Angular and XSS In A Nutshell

Input:

```
const html = '<img src=x onerror=alert(1)/>';
```

Angular sanitizes this!

```
<span [innerHTML]="html"></span>
```

```
<img src=x />
```

Input:

```
const link = 'javascript:alert(1)';
```

Angular sanitizes this!

```
<a [href]="link">Click</a>
```

```
unsafe:javascript:alert(1)
```

Mitigation Tracking: Angular and XSS

Vulnerability name	Cross-Site Scripting (XSS)
Threat type (C/I/A/N)	C/I/-/N
Qualitative severity	Medium to high
How does technology help?	Angular does automatic HTML encoding by default and comes with a sanitizer for `[href]` and `[innerHTML]` [1].
What are edge cases?	When `bypassSecurityTrust*` is used [2].
Automated checks	- We use SAST to disallow `bypassSecurityTrust*` [3] - We use a Linter to disallow DOM XSS sinks [4]
Defense-in-depth measures	We use a strong Content Security Policy [5].
Residual risk	Developers use insecure DOM APIs [6] directly.
Links and references	*Links above, internal policies and guidelines, requirements documents, …*

Learning Resources

Where to learn more about secure software development

Understanding is Key

You won't get software security for free

- Cultivate a culture of continuous learning
- Understand your language, runtime, platform, IDE, build tools, relevant vulnerability classes, threats
- Develop by-design countermeasures
- Simplify and reduce

Where Can I Learn Software Security?

- **General**
  - Testing and securing your own software!
  - sec4dev 😉
  - Security Meetup by SBA Research 😉

Where Can I Learn Software Security?

- **Web Security**
  - OWASP Juice Shop
  - PortSwigger Web Academy
  - OWASP Resources
  - OWASP Application Security Verification Standard (ASVS)

Wrap-up

Key take-aways

Wrap-up

1. Most companies are software companies
2. Initial velocity often goes to the cost of sustained velocity
3. Secure software can be created with any language, but knowing the security properties still helps
4. Use OWASP SAMM to widen your view on secure development
5. Internalize secure coding practices as a base
6. Clean code tends to be testable and secure code
7. Dependency checks are a good first step towards automation
8. **Deep understanding of the software, vulnerability classes and threats is key to securing your software**

SBA Research

Bridging Science and Industry

Contact us: anfragen@sba-research.org

Software Security Services

Penetration Testing | Architecture Reviews | Threat Modeling | Secure SDLC Analysis | Security Champion Programs | DevSecOps and Security Automation | Cloud Security

Applied Research

Industrial Security | IIoT Security | Mathematics for Security Research | Machine Learning | Blockchain | Network Security | Sustainable Software Systems | Usable Security

Knowledge Transfer

SBA Live Academy | sec4dev | Trainings | Events | Teaching | sbaPRIME

SBA Research

Sponsored by

MICRO FOCUS®

Have a look at the Micro Focus booth and win a 3D Printer!


Photo by Emily Morter on Unsplash
Follow me on Twitter! @_thomaskonrad

Competence Centers for
Excellent Technologies

Thomas Konrad

SBA Research

Floragasse 7, 1040 Vienna

+43 664 889 272 17

tkonrad@sba-research.org

Bundesministerium Klimaschutz, Umwelt, Energie, Mobilität, Innovation und Technologie
Bundesministerium Digitalisierung und Wirtschaftsstandort
FFG Forschung wirkt.
wirtschafts agentur wien Ein Fonds der Stadt Wien
European Commission
FWF Der Wissenschaftsfonds.
netidee OPEN INNOVATIONS