Security in our code review? Check!

Daniel Zollinger
Security Consultant

@dznz
daniel@safestack.io

Kia ora koutou

We’re on the final stretch!

Is a Code Review a Security Control?

Code reviews are already hard

- It’s a natural bottleneck
- Entrenched power dynamics
- We identify with our work
  … or at least want to do a good job!
- Every team’s culture is different

Your rules can be gamed

- Test line coverage **must** be 85% or higher
- Some lines of code are not usefully tested
- Exclude untestable classes from total lines

… 100% coverage!

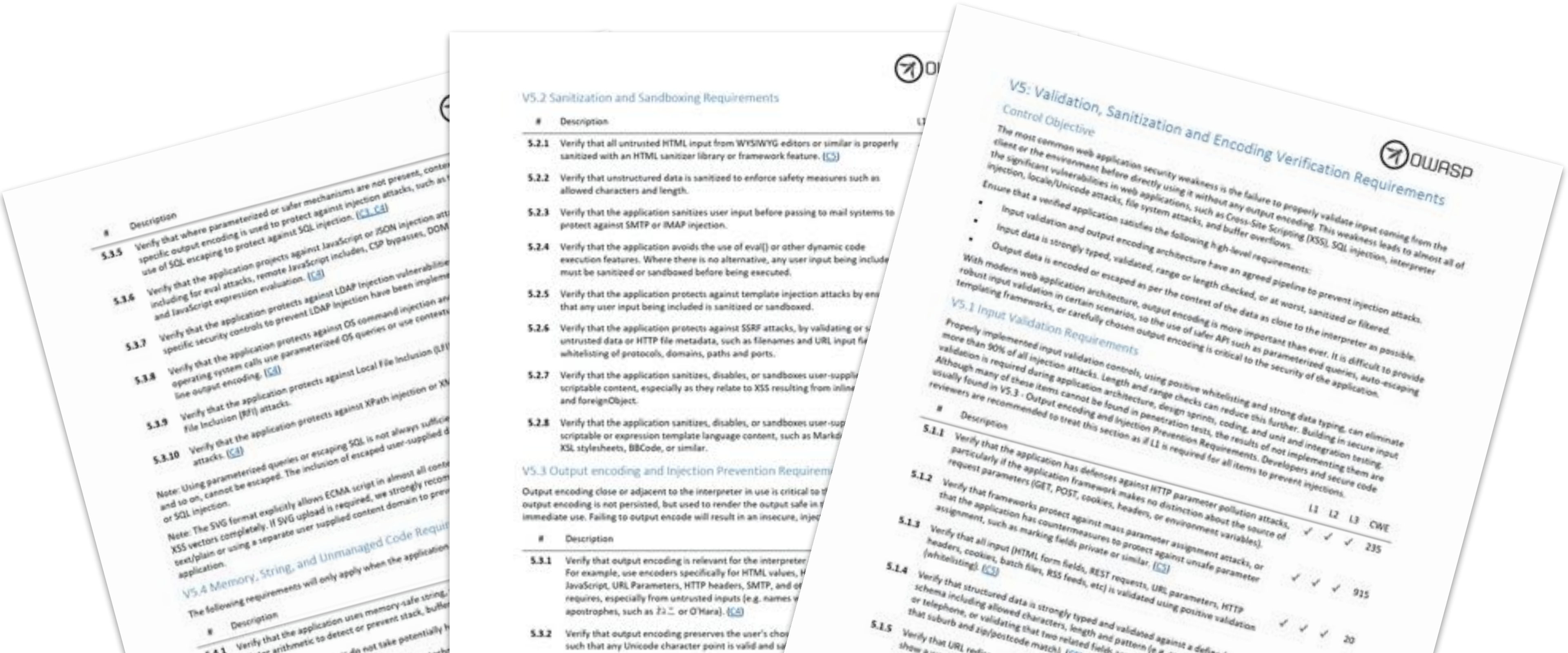
What can I ~~steal~~ borrow?

Application Security Verification Standard (2019)

OWASP

Table of Contents

Application Security Verification Standard (2019)



#	Description
5.3.5	Verify that where parameterized or safer mechanisms are not present, conte specific output encoding is used to protect against injection attacks, such as use of SQL escaping to protect against SQL injection. (C3, C4)
5.3.6	Verify that the application projects against JavaScript or JSON injection att including for eval attacks, remote JavaScript includes, CSP bypasses, DOM and JavaScript expression evaluation. (C4)
5.3.7	Verify that the application protects against LDAP Injection vulnerabilitie specific security controls to prevent LDAP Injection have been impleme
5.3.8	Verify that the application protects against OS command injection an operating system calls use parameterized OS queries or use contextu line output encoding. (C4)
5.3.9	Verify that the application protects against Local File Inclusion (LFI) File Inclusion (RFI) attacks.
5.3.10	Verify that the application protects against XPath injection or XM attacks. (C4)

Note: Using parameterized queries or escaping SQL is not always suffici and so on, cannot be escaped. The inclusion of escaped user-supplied d or SQL injection.

Note: The SVG format explicitly allows ECMA script in almost all conte XSS vectors completely. If SVG upload is required, we strongly recom text/plain or using a separate user supplied content domain to prev application.

V5.4 Memory, String, and Unmanaged Code Requir

The following requirements will only apply when the application

#	Description
5.4.1	Verify that the application uses memory-safe string, ar arithmetic to detect or prevent stack, buffe do not take potentially h

V5.2 Sanitization and Sandboxing Requirements

#	Description	L1
5.2.1	Verify that all untrusted HTML input from WYSIWYG editors or similar is properly sanitized with an HTML sanitizer library or framework feature. (C5)	
5.2.2	Verify that unstructured data is sanitized to enforce safety measures such as allowed characters and length.	
5.2.3	Verify that the application sanitizes user input before passing to mail systems to protect against SMTP or IMAP injection.	
5.2.4	Verify that the application avoids the use of eval() or other dynamic code execution features. Where there is no alternative, any user input being include must be sanitized or sandboxed before being executed.	
5.2.5	Verify that the application protects against template injection attacks by ens that any user input being included is sanitized or sandboxed.	
5.2.6	Verify that the application protects against SSRF attacks, by validating or s untrusted data or HTTP file metadata, such as filenames and URL input fie whitelisting of protocols, domains, paths and ports.	
5.2.7	Verify that the application sanitizes, disables, or sandboxes user-suppli scriptable content, especially as they relate to XSS resulting from inline and foreignObject.	
5.2.8	Verify that the application sanitizes, disables, or sandboxes user-sup scriptable or expression template language content, such as Markd XSL stylesheets, BBCode, or similar.	

V5.3 Output encoding and Injection Prevention Requirem

Output encoding close or adjacent to the interpreter in use is critical to t output encoding is not persisted, but used to render the output safe in immediate use. Failing to output encode will result in an insecure, injec

#	Description
5.3.1	Verify that output encoding is relevant for the interpreter For example, use encoders specifically for HTML values, H JavaScript, URL Parameters, HTTP headers, SMTP, and ot requires, especially from untrusted inputs (e.g. names w apostrophes, such as ねこ or O'Hara). (C4)
5.3.2	Verify that output encoding preserves the user's cho such that any Unicode character point is valid and sa

V5: Validation, Sanitization and Encoding Verification Requirements

Control Objective

The most common web application security weakness is the failure to properly validate input coming from the client or the environment before directly using it without any output encoding. This weakness leads to almost all of the significant vulnerabilities in web applications, such as Cross-Site Scripting (XSS), SQL injection, interpreter injection, locale/Unicode attacks, file system attacks, and buffer overflows.

Ensure that a verified application satisfies the following high-level requirements:

- Input validation and output encoding architecture have an agreed pipeline to prevent injection attacks.
- Input data is strongly typed, validated, range or length checked, or at worst, sanitized or filtered.
- Output data is encoded or escaped as per the context of the data as close to the interpreter as possible.

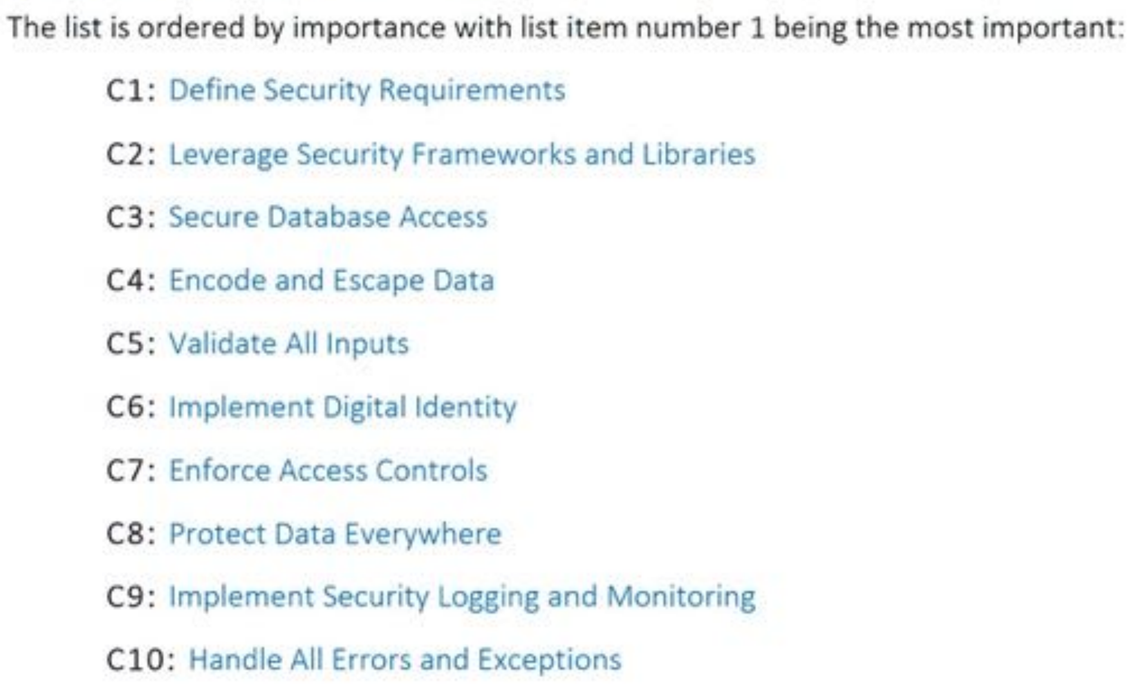
With modern web application architecture, output encoding is more important than ever. It is difficult to provide robust input validation in certain scenarios, so the use of safer API such as parameterized queries, auto-escaping templating frameworks, or carefully chosen output encoding is critical to the security of the application.

V5.1 Input Validation Requirements

Properly implemented input validation controls, using positive whitelisting and strong data typing, can eliminate more than 90% of all injection attacks. Length and range checks can reduce this further. Building in secure input validation is required during application architecture, design sprints, coding, and unit and integration testing. Although many of these items cannot be found in penetration tests, the results of not implementing them are usually found in V5.3 - Output encoding and Injection Prevention Requirements. Developers and secure code reviewers are recommended to treat this section as if L1 is required for all items to prevent injections.

#	Description	L1	L2	L3	CWE
5.1.1	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, or environment variables).	✓	✓	✓	235
5.1.2	Verify that frameworks protect against mass parameter assignment attacks, or that the application has countermeasures to protect against unsafe parameter assignment, such as marking fields private or similar. (C5)	✓	✓	✓	915
5.1.3	Verify that all input (HTML form fields, REST requests, URL parameters, HTTP headers, cookies, batch files, RSS feeds, etc) is validated using positive validation (whitelisting). (C5)	✓	✓	✓	20
5.1.4	Verify that structured data is strongly typed and validated against a defin schema including allowed characters, length and pattern (e.g. or telephone, or validating that two related fields that suburb and zip/postcode match). (C5)				
5.1.5	Verify that URL redi show

OWASP Top 10 Proactive Controls (2018)



The list is ordered by importance with list item number 1 being the most important:

C1: Define Security Requirements

C2: Leverage Security Frameworks and Libraries

C3: Secure Database Access

C4: Encode and Escape Data

C5: Validate All Inputs

C6: Implement Digital Identity

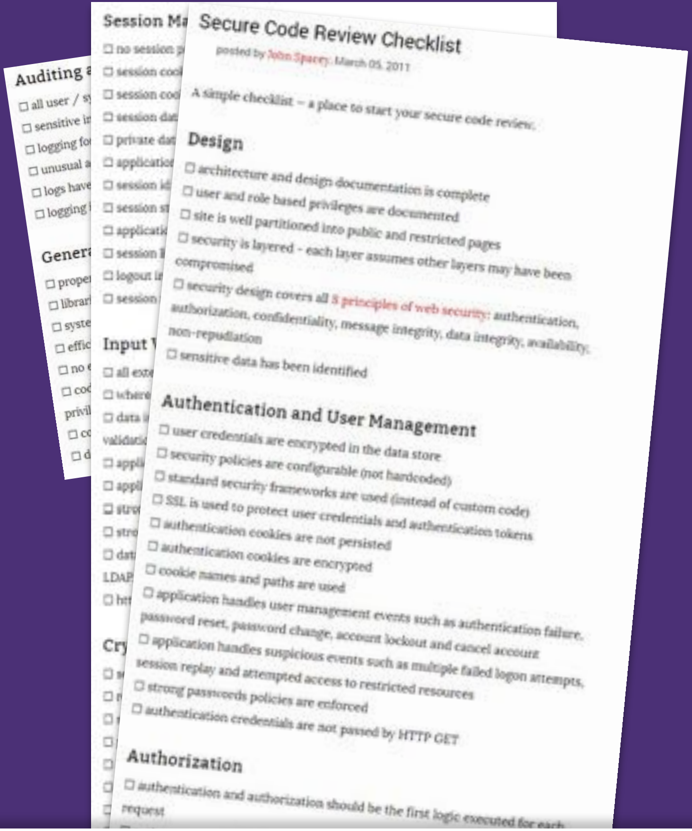
C7: Enforce Access Controls

C8: Protect Data Everywhere

C9: Implement Security Logging and Monitoring

C10: Handle All Errors and Exceptions

Source: https://owasp.org/www-project-proactive-controls/



Secure Code Review Checklist

posted by John Sparey, March 05, 2011

A simple checklist – a place to start your secure code review.

Design

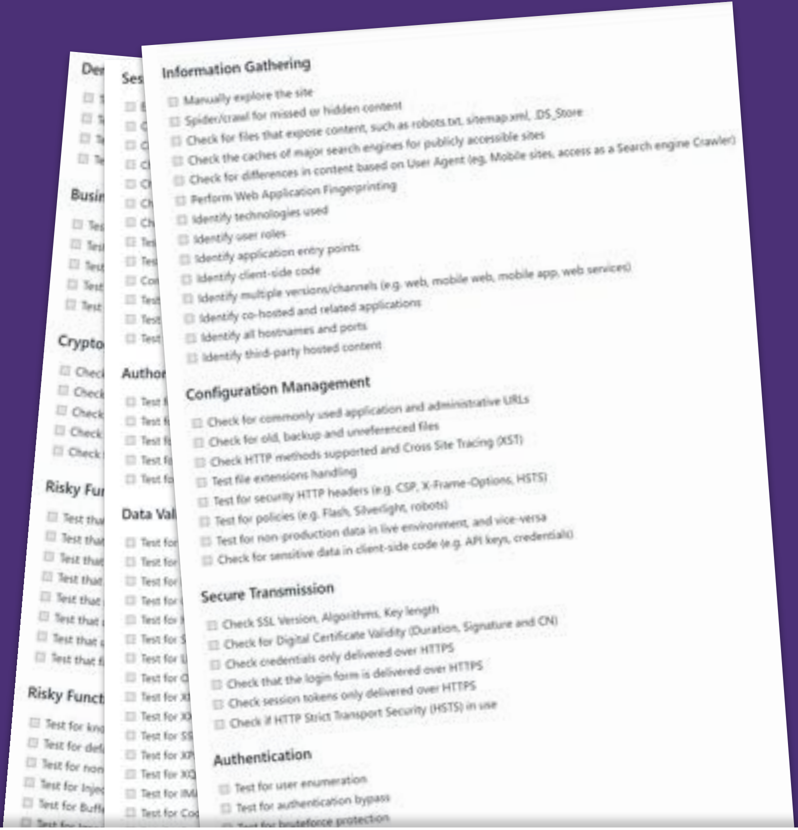
- ☐ architecture and design documentation is complete
- ☐ user and role based privileges are documented
- ☐ site is well partitioned into public and restricted pages
- ☐ security is layered – each layer assumes other layers may have been compromised
- ☐ security design covers all 8 principles of web security: authentication, authorization, confidentiality, message integrity, data integrity, availability, non-repudiation
- ☐ sensitive data has been identified

Authentication and User Management

- ☐ user credentials are encrypted in the data store
- ☐ security policies are configurable (not hardcoded)
- ☐ standard security frameworks are used (instead of custom code)
- ☐ SSL is used to protect user credentials and authentication tokens
- ☐ authentication cookies are not persisted
- ☐ authentication cookies are encrypted
- ☐ cookie names and paths are used
- ☐ application handles user management events such as authentication failure, password reset, password change, account lockout and cancel account
- ☐ application handles suspicious events such as multiple failed logon attempts, session replay and attempted access to restricted resources
- ☐ strong passwords policies are enforced
- ☐ authentication credentials are not passed by HTTP GET

Authorization

- ☐ authentication and authorization should be the first logic executed for each request



Information Gathering

- ☐ Manually explore the site
- ☐ Spider/crawl for missed or hidden content
- ☐ Check for files that expose content, such as robots.txt, sitemap.xml, .DS_Store
- ☐ Check the caches of major search engines for publicly accessible sites
- ☐ Check for differences in content based on User Agent (eg, Mobile sites, access as a Search engine Crawler)
- ☐ Perform Web Application Fingerprinting
- ☐ Identify technologies used
- ☐ Identify user roles
- ☐ Identify application entry points
- ☐ Identify client-side code
- ☐ Identify multiple versions/channels (e.g. web, mobile web, mobile app, web services)
- ☐ Identify co-hosted and related applications
- ☐ Identify all hostnames and ports
- ☐ Identify third-party hosted content

Configuration Management

- ☐ Check for commonly used application and administrative URLs
- ☐ Check for old, backup and unreferenced files
- ☐ Check HTTP methods supported and Cross Site Tracing (XST)
- ☐ Test file extensions handling
- ☐ Test for security HTTP headers (e.g. CSP, X-Frame-Options, HSTS)
- ☐ Test for policies (e.g. Flash, Silverlight, robots)
- ☐ Test for non-production data in live environment, and vice-versa
- ☐ Check for sensitive data in client-side code (e.g. API keys, credentials)

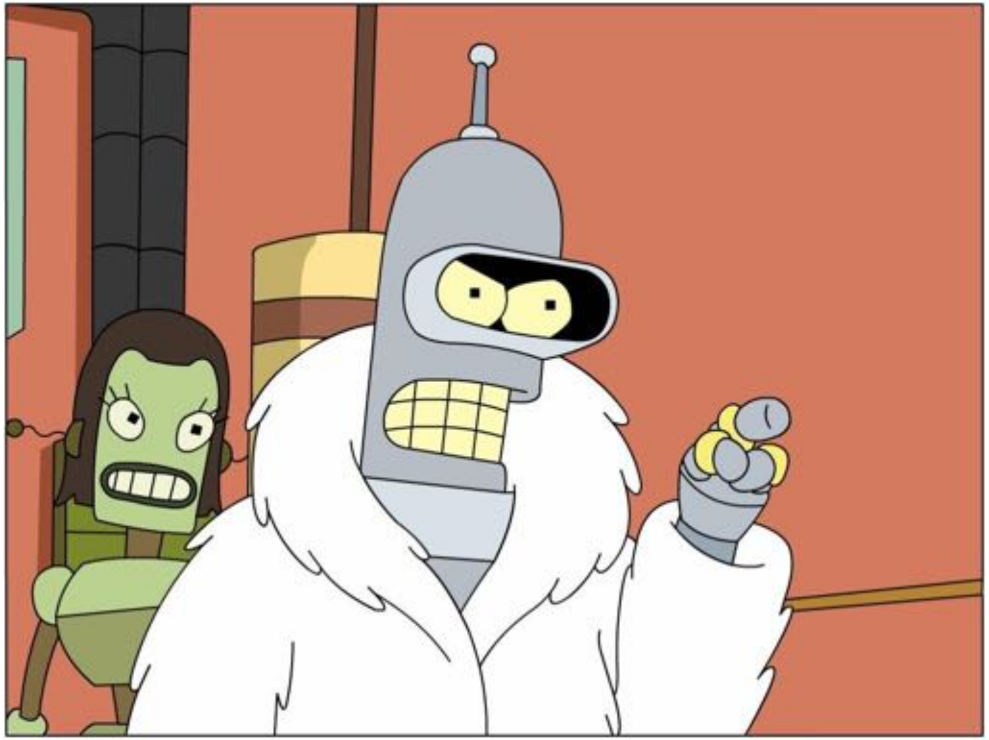
Secure Transmission

- ☐ Check SSL Version, Algorithms, Key length
- ☐ Check for Digital Certificate Validity (Duration, Signature and CN)
- ☐ Check credentials only delivered over HTTPS
- ☐ Check that the login form is delivered over HTTPS
- ☐ Check session tokens only delivered over HTTPS
- ☐ Check if HTTP Strict Transport Security (HSTS) in use

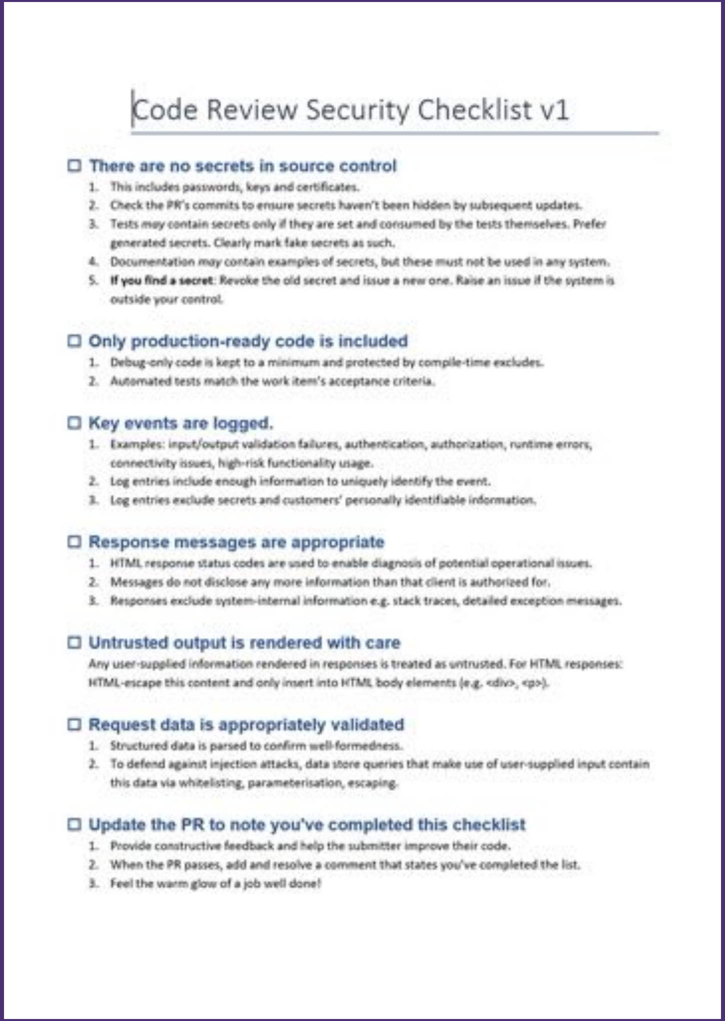
Authentication

- ☐ Test for user enumeration
- ☐ Test for authentication bypass
- ☐ Test for bruteforce protection

Other checklists on the internet



Code Review Security Checklist v1



Code Review Security Checklist v1

☐ There are no secrets in source control

1. This includes passwords, keys and certificates.
2. Check the PR's commits to ensure secrets haven't been hidden by subsequent updates.
3. Tests may contain secrets only if they are set and consumed by the tests themselves. Prefer generated secrets. Clearly mark fake secrets as such.
4. Documentation may contain examples of secrets, but these must not be used in any system.
5. **If you find a secret**: Revoke the old secret and issue a new one. Raise an issue if the system is outside your control.

☐ Only production-ready code is included

1. Debug-only code is kept to a minimum and protected by compile-time excludes.
2. Automated tests match the work item's acceptance criteria.

☐ Key events are logged.

1. Examples: input/output validation failures, authentication, authorization, runtime errors, connectivity issues, high-risk functionality usage.
2. Log entries include enough information to uniquely identify the event.
3. Log entries exclude secrets and customers' personally identifiable information.

☐ Response messages are appropriate

1. HTML response status codes are used to enable diagnosis of potential operational issues.
2. Messages do not disclose any more information than that client is authorized for.
3. Responses exclude system-internal information e.g. stack traces, detailed exception messages.

☐ Untrusted output is rendered with care

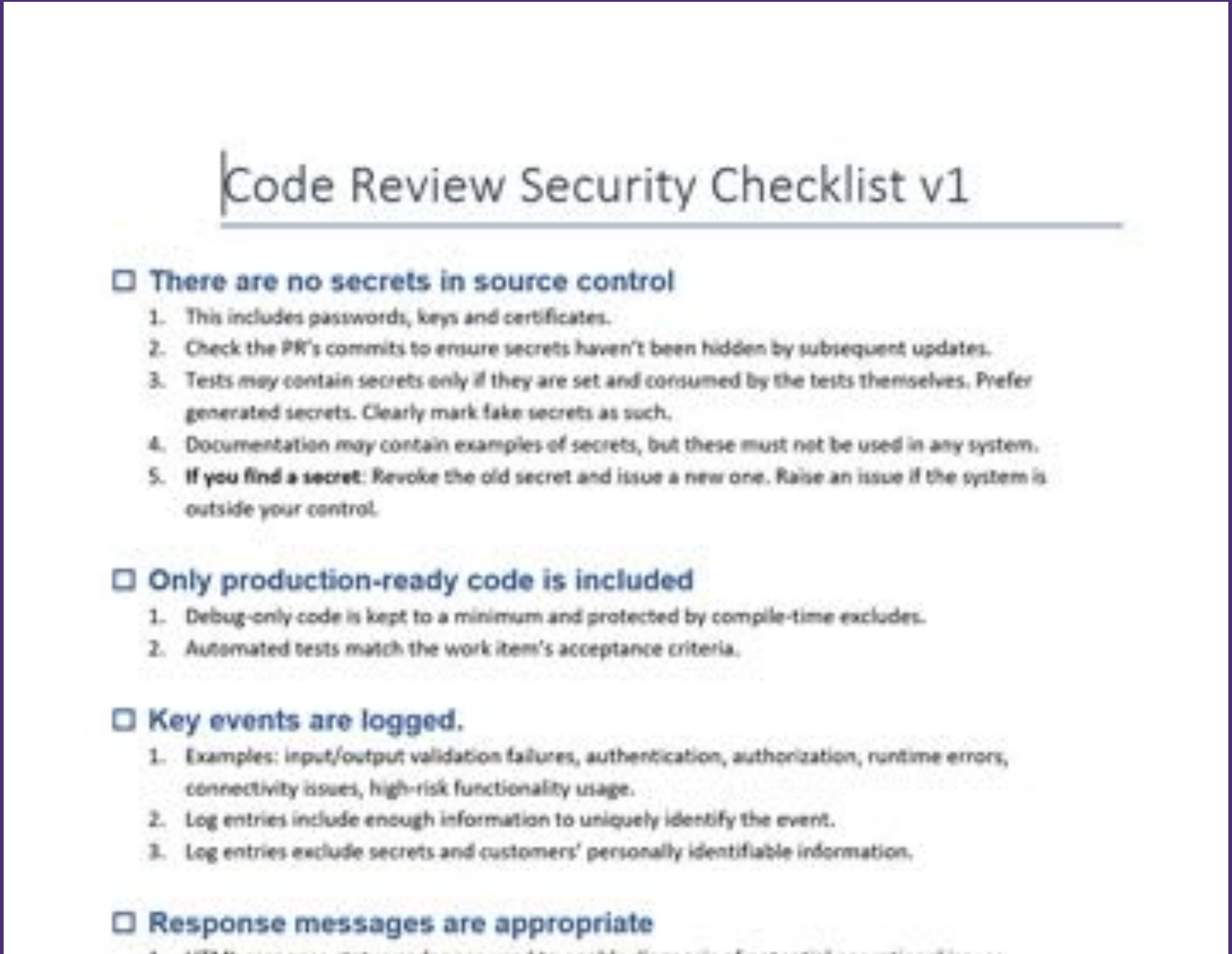
Any user-supplied information rendered in responses is treated as untrusted. For HTML responses: HTML-escape this content and only insert into HTML body elements (e.g. <div>, <p>).

☐ Request data is appropriately validated

1. Structured data is parsed to confirm well-formedness.
2. To defend against injection attacks, data store queries that make use of user-supplied input contain this data via whitelisting, parameterisation, escaping.

☐ Update the PR to note you've completed this checklist

1. Provide constructive feedback and help the submitter improve their code.
2. When the PR passes, add and resolve a comment that states you've completed the list.
3. Feel the warm glow of a job well done!



Code Review Security Checklist v1

☐ There are no secrets in source control

1. This includes passwords, keys and certificates.
2. Check the PR's commits to ensure secrets haven't been hidden by subsequent updates.
3. Tests may contain secrets only if they are set and consumed by the tests themselves. Prefer generated secrets. Clearly mark fake secrets as such.
4. Documentation may contain examples of secrets, but these must not be used in any system.
5. **If you find a secret**: Revoke the old secret and issue a new one. Raise an issue if the system is outside your control.

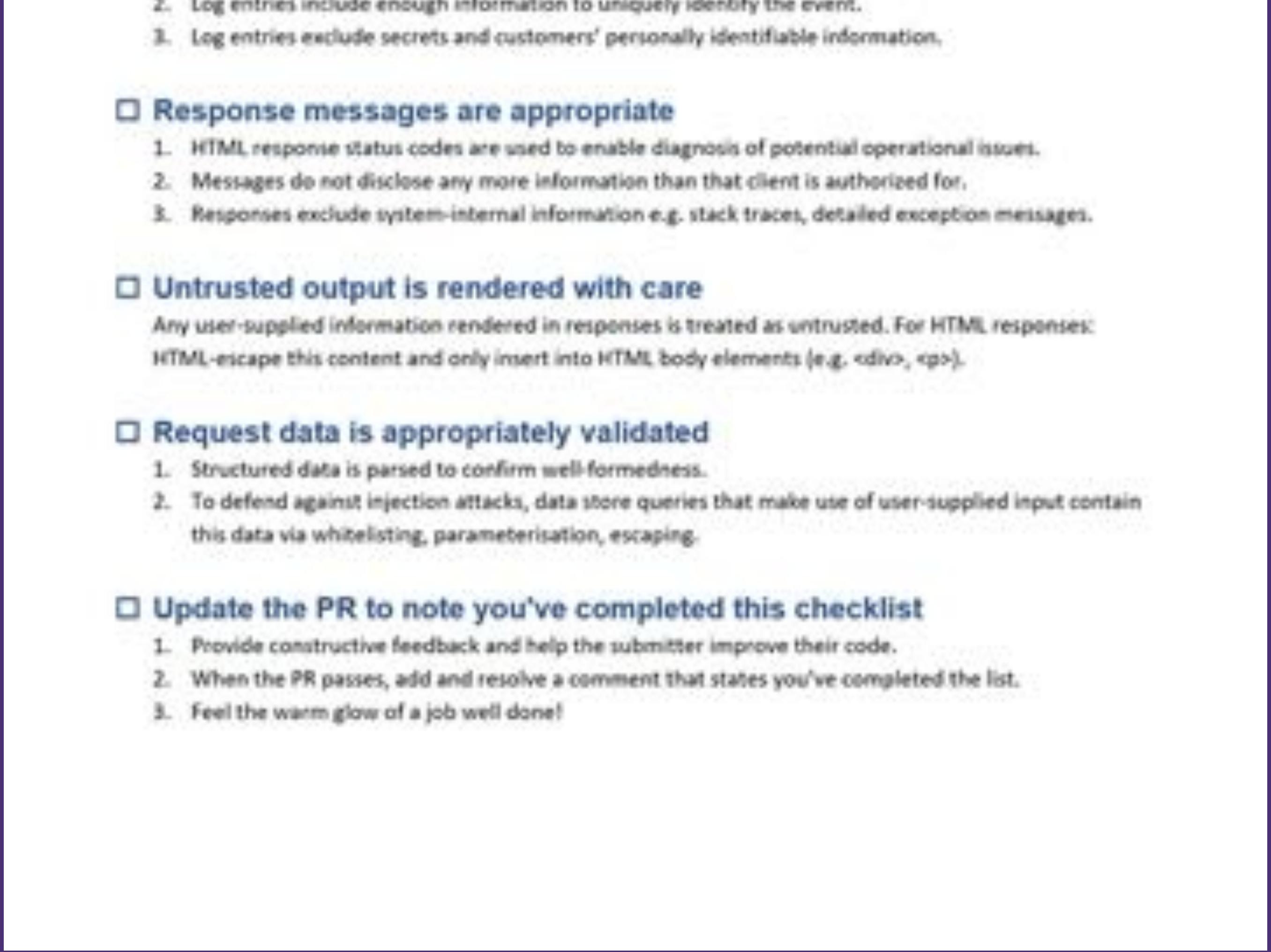
☐ Only production-ready code is included

1. Debug-only code is kept to a minimum and protected by compile-time excludes.
2. Automated tests match the work item's acceptance criteria.

☐ Key events are logged.

1. Examples: input/output validation failures, authentication, authorization, runtime errors, connectivity issues, high-risk functionality usage.
2. Log entries include enough information to uniquely identify the event.
3. Log entries exclude secrets and customers' personally identifiable information.

☐ Response messages are appropriate



2. Log entries include enough information to uniquely identify the event.
3. Log entries exclude secrets and customers' personally identifiable information.

☐ Response messages are appropriate

1. HTML response status codes are used to enable diagnosis of potential operational issues.
2. Messages do not disclose any more information than that client is authorized for.
3. Responses exclude system-internal information e.g. stack traces, detailed exception messages.

☐ Untrusted output is rendered with care

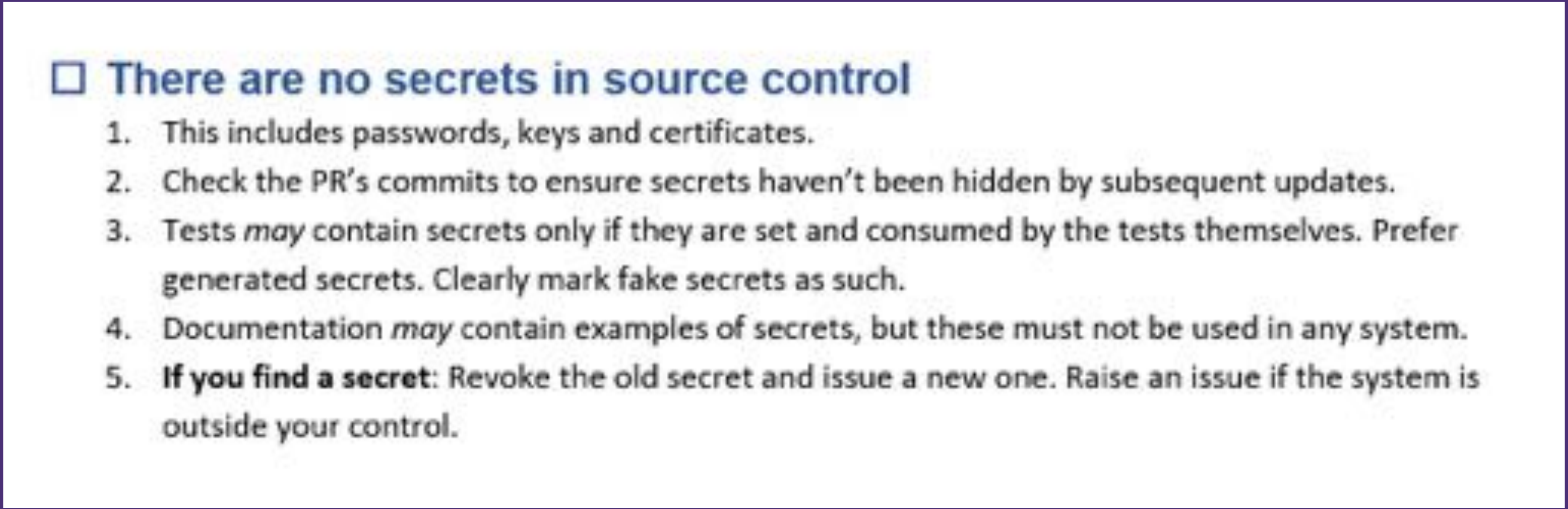
Any user-supplied information rendered in responses is treated as untrusted. For HTML responses: HTML-escape this content and only insert into HTML body elements (e.g. <div>, <p>).

☐ Request data is appropriately validated

1. Structured data is parsed to confirm well-formedness.
2. To defend against injection attacks, data store queries that make use of user-supplied input contain this data via whitelisting, parameterisation, escaping.

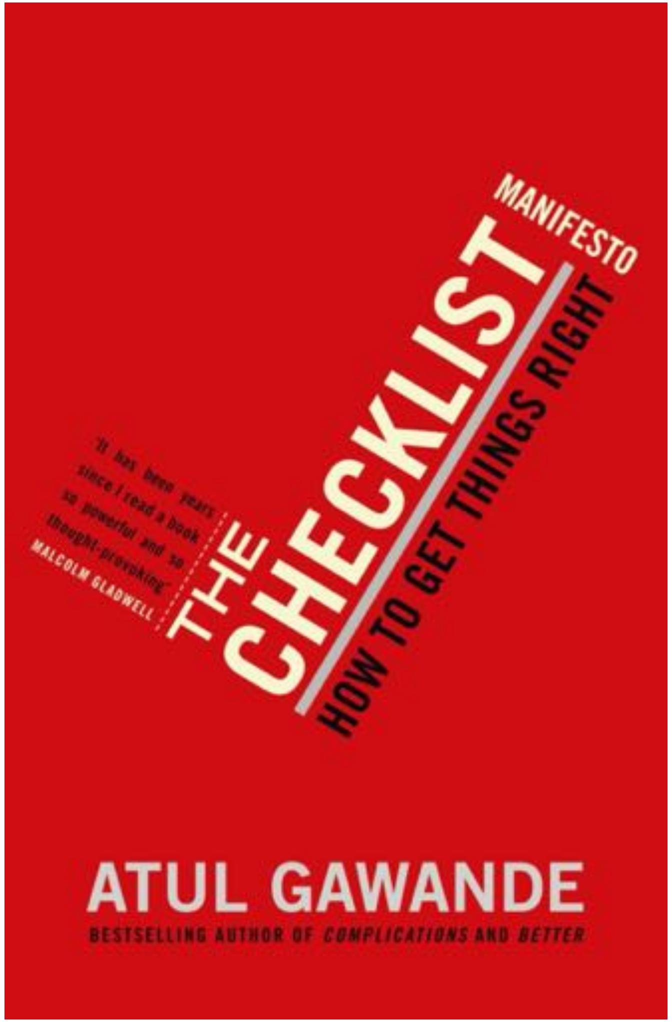
☐ Update the PR to note you've completed this checklist

1. Provide constructive feedback and help the submitter improve their code.
2. When the PR passes, add and resolve a comment that states you've completed the list.
3. Feel the warm glow of a job well done!



☐ There are no secrets in source control

1. This includes passwords, keys and certificates.
2. Check the PR's commits to ensure secrets haven't been hidden by subsequent updates.
3. Tests *may* contain secrets only if they are set and consumed by the tests themselves. Prefer generated secrets. Clearly mark fake secrets as such.
4. Documentation *may* contain examples of secrets, but these must not be used in any system.
5. **If you find a secret**: Revoke the old secret and issue a new one. Raise an issue if the system is outside your control.



THE CHECKLIST MANIFESTO

HOW TO GET THINGS RIGHT

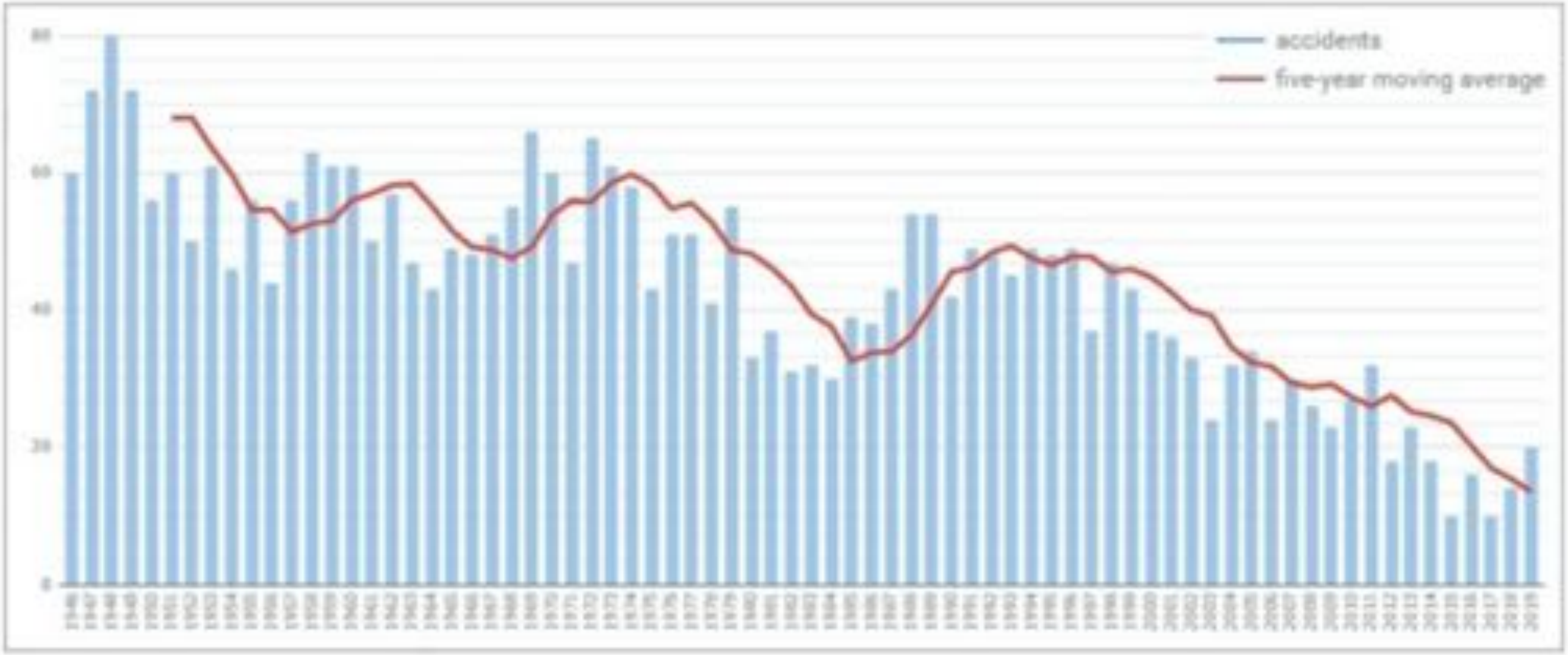
'It has been years since I read a book so powerful and so thought-provoking'
MALCOLM GLADWELL

ATUL GAWANDE

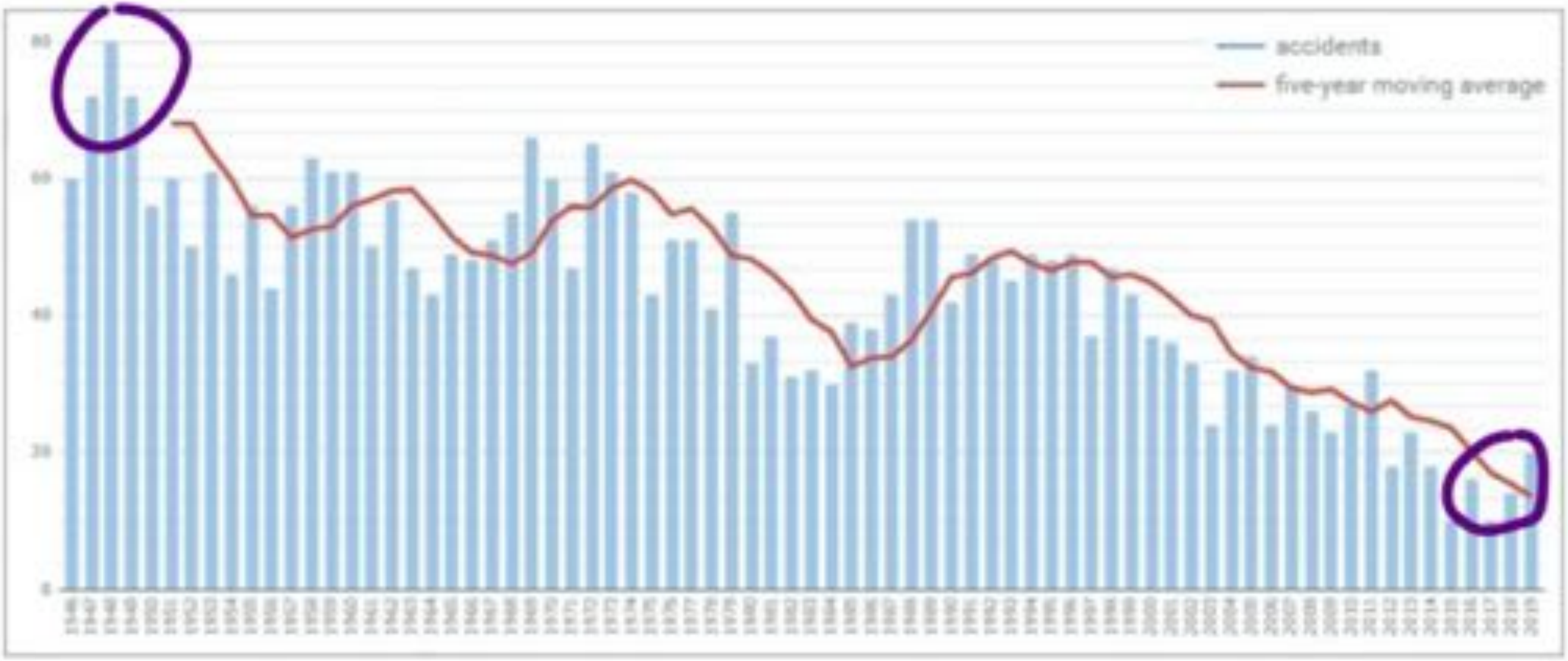
BESTSELLING AUTHOR OF *COMPLICATIONS* AND *BETTER*

B-17 Checklist (1935)

Aviation Safety (1946-2019)



Aviation Safety (1946-2019)

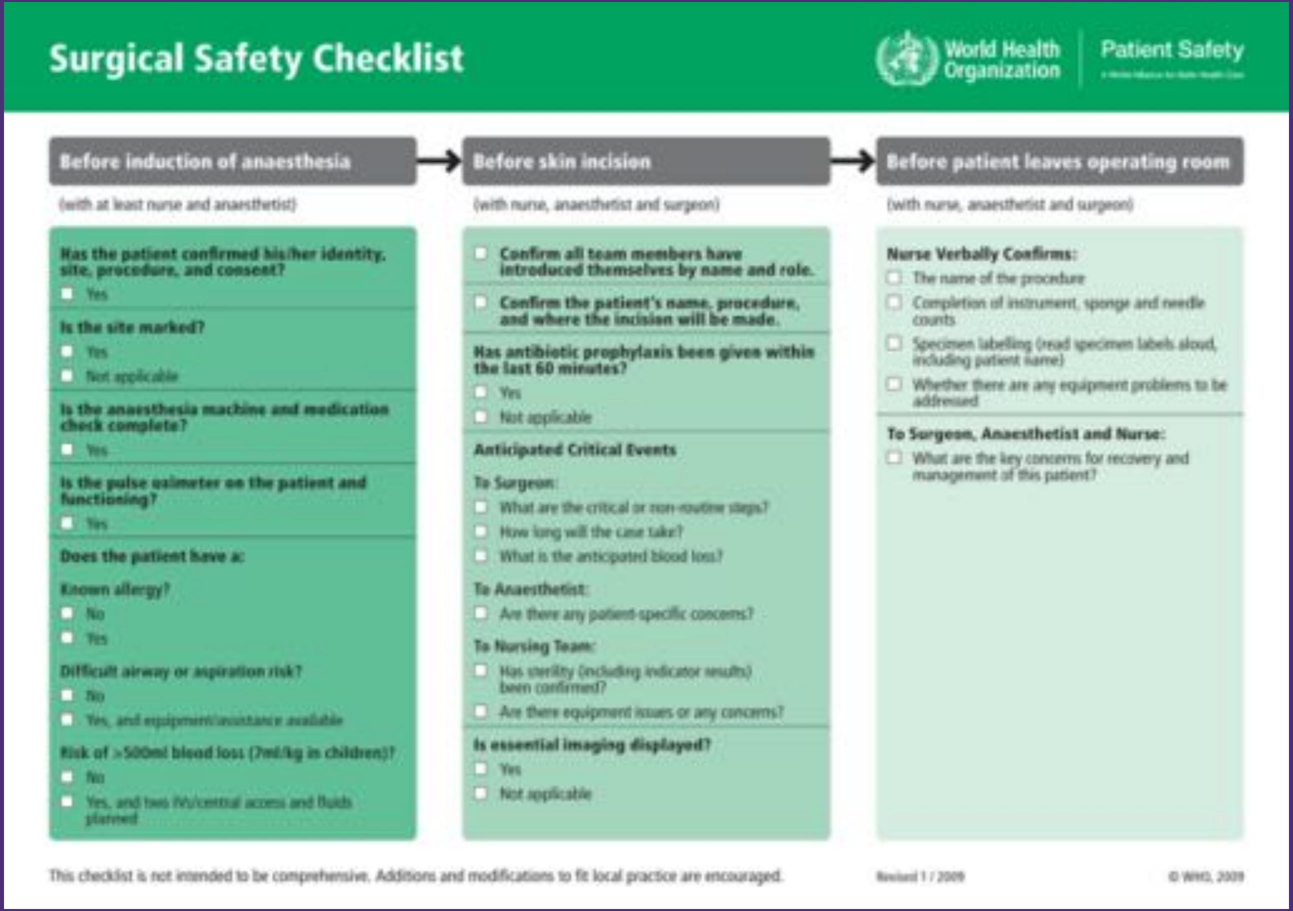


2001	Peter Pronovost creates central line infection checklist **Line injections drop from 11% to 0%**
2006	Atul Gawande & WHO begin checklist project
2009	Results released in New England Journal of Medicine: **36% drop in complications** **47% drop in deaths**

WHO Surgical Safety Checklist

Mandates + feedback	> 26% drop in deaths
South Carolina Mandate + Training + Feedback	22% drop in deaths
Veterans Health Administration Mandates + team training	18% drop in deaths
Ontario - Mandate only	0% drop in deaths

WHO SSC - 10 Years On



Surgical Safety Checklist

World Health Organization | Patient Safety

Before induction of anaesthesia	Before skin incision	Before patient leaves operating room
(with at least nurse and anaesthetist)	(with nurse, anaesthetist and surgeon)	(with nurse, anaesthetist and surgeon)

Before induction of anaesthesia

Has the patient confirmed his/her identity, site, procedure, and consent?
- ☐ Yes

Is the site marked?
- ☐ Yes
- ☐ Not applicable

Is the anaesthesia machine and medication check complete?
- ☐ Yes

Is the pulse oximeter on the patient and functioning?
- ☐ Yes

Does the patient have a:

Known allergy?
- ☐ No
- ☐ Yes

Difficult airway or aspiration risk?
- ☐ No
- ☐ Yes, and equipment/assistance available

Risk of >500ml blood loss (7ml/kg in children)?
- ☐ No
- ☐ Yes, and two IVs/central access and fluids planned

Before skin incision

- ☐ **Confirm all team members have introduced themselves by name and role.**
- ☐ **Confirm the patient's name, procedure, and where the incision will be made.**

Has antibiotic prophylaxis been given within the last 60 minutes?
- ☐ Yes
- ☐ Not applicable

Anticipated Critical Events

To Surgeon:
- ☐ What are the critical or non-routine steps?
- ☐ How long will the case take?
- ☐ What is the anticipated blood loss?

To Anaesthetist:
- ☐ Are there any patient-specific concerns?

To Nursing Team:
- ☐ Has sterility (including indicator results) been confirmed?
- ☐ Are there equipment issues or any concerns?

Is essential imaging displayed?
- ☐ Yes
- ☐ Not applicable

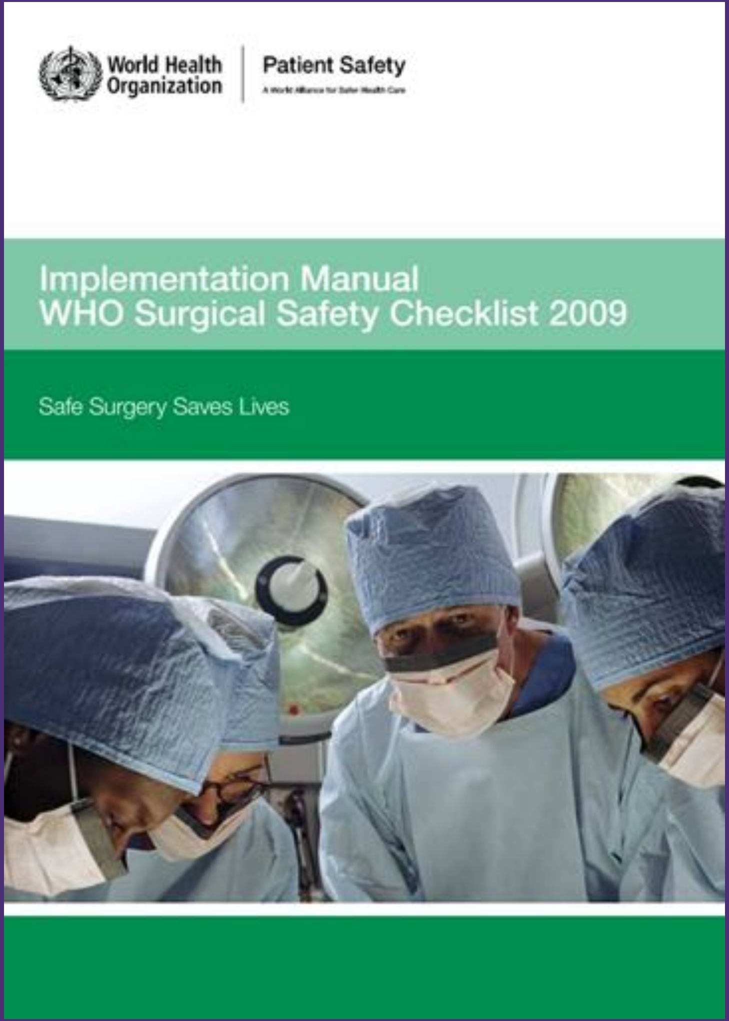
Before patient leaves operating room

Nurse Verbally Confirms:
- ☐ The name of the procedure
- ☐ Completion of instrument, sponge and needle counts
- ☐ Specimen labelling (read specimen labels aloud, including patient name)
- ☐ Whether there are any equipment problems to be addressed

To Surgeon, Anaesthetist and Nurse:
- ☐ What are the key concerns for recovery and management of this patient?

This checklist is not intended to be comprehensive. Additions and modifications to fit local practice are encouraged. Revised 1 / 2009 © WHO, 2009

Source:
https://www.who.int/patientsafety/safesurgery/checklist/en/

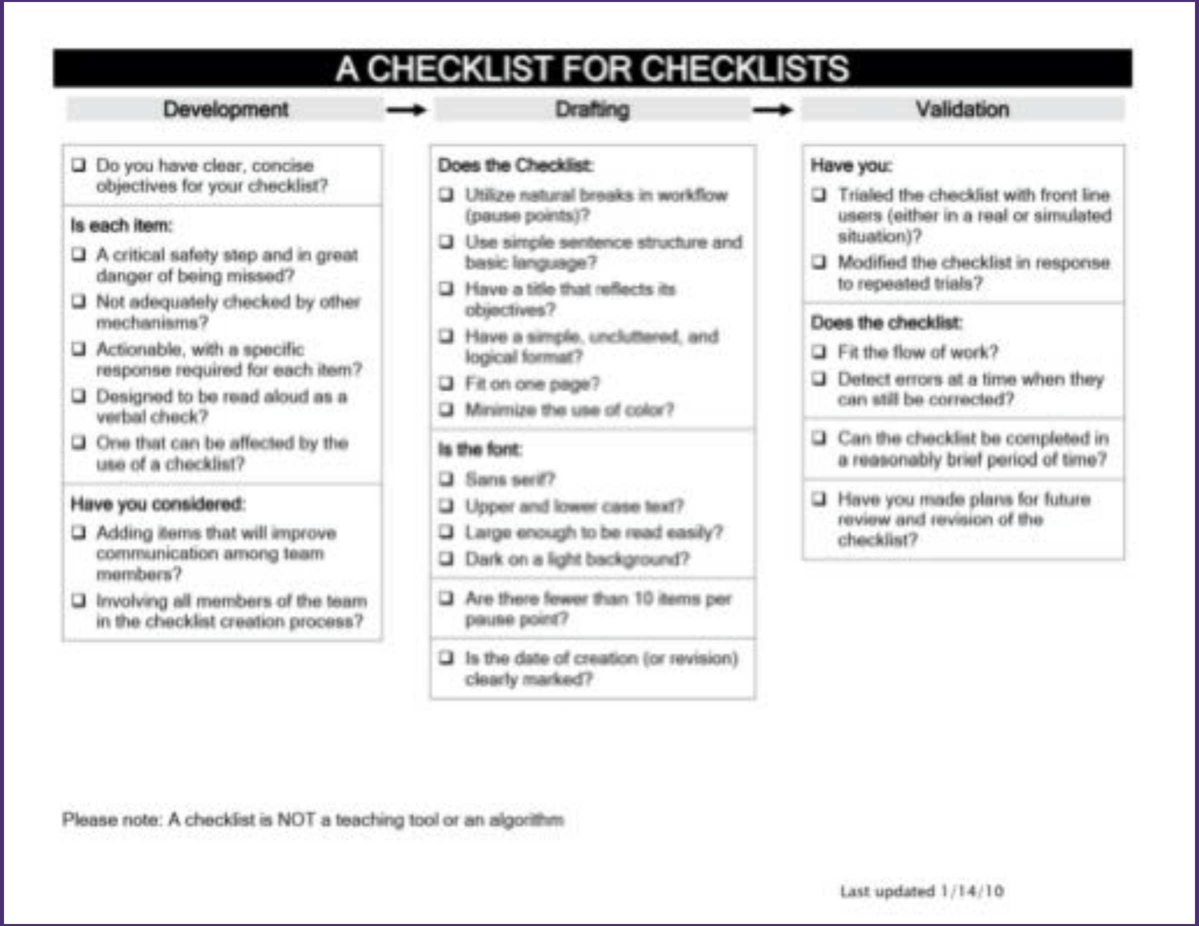


World Health Organization | Patient Safety

A World Alliance for Safer Health Care

Implementation Manual
WHO Surgical Safety Checklist 2009

Safe Surgery Saves Lives



A CHECKLIST FOR CHECKLISTS

Development → **Drafting** → **Validation**

Development

- ❑ Do you have clear, concise objectives for your checklist?

Is each item:

- ❑ A critical safety step and in great danger of being missed?
- ❑ Not adequately checked by other mechanisms?
- ❑ Actionable, with a specific response required for each item?
- ❑ Designed to be read aloud as a verbal check?
- ❑ One that can be affected by the use of a checklist?

Have you considered:

- ❑ Adding items that will improve communication among team members?
- ❑ Involving all members of the team in the checklist creation process?

Drafting

Does the Checklist:

- ❑ Utilize natural breaks in workflow (pause points)?
- ❑ Use simple sentence structure and basic language?
- ❑ Have a title that reflects its objectives?
- ❑ Have a simple, uncluttered, and logical format?
- ❑ Fit on one page?
- ❑ Minimize the use of color?

Is the font:

- ❑ Sans serif?
- ❑ Upper and lower case text?
- ❑ Large enough to be read easily?
- ❑ Dark on a light background?

- ❑ Are there fewer than 10 items per pause point?

- ❑ Is the date of creation (or revision) clearly marked?

Validation

Have you:

- ❑ Trialed the checklist with front line users (either in a real or simulated situation)?
- ❑ Modified the checklist in response to repeated trials?

Does the checklist:

- ❑ Fit the flow of work?
- ❑ Detect errors at a time when they can still be corrected?

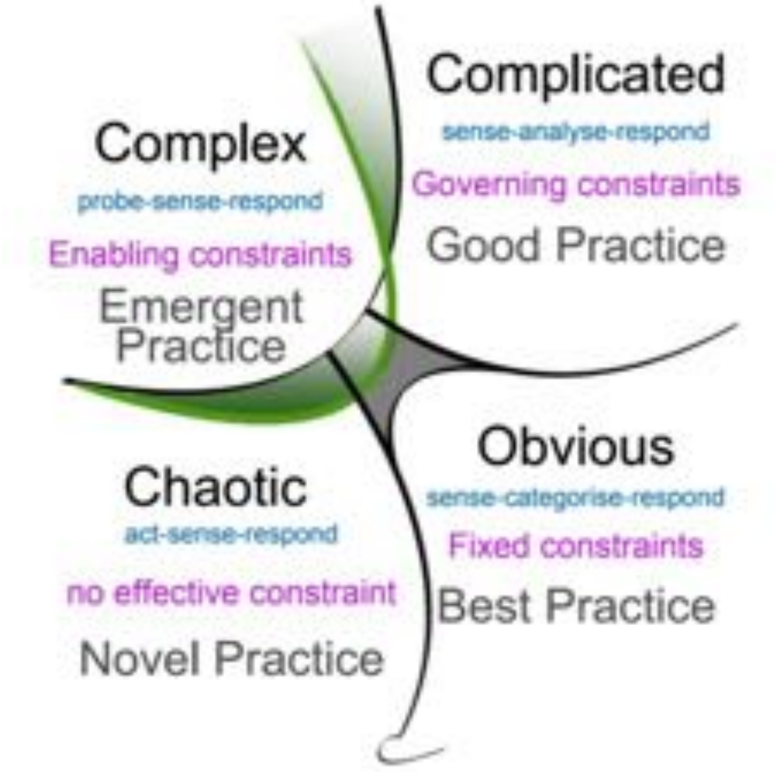
- ❑ Can the checklist be completed in a reasonably brief period of time?

- ❑ Have you made plans for future review and revision of the checklist?

Please note: A checklist is NOT a teaching tool or an algorithm

Last updated 1/14/10

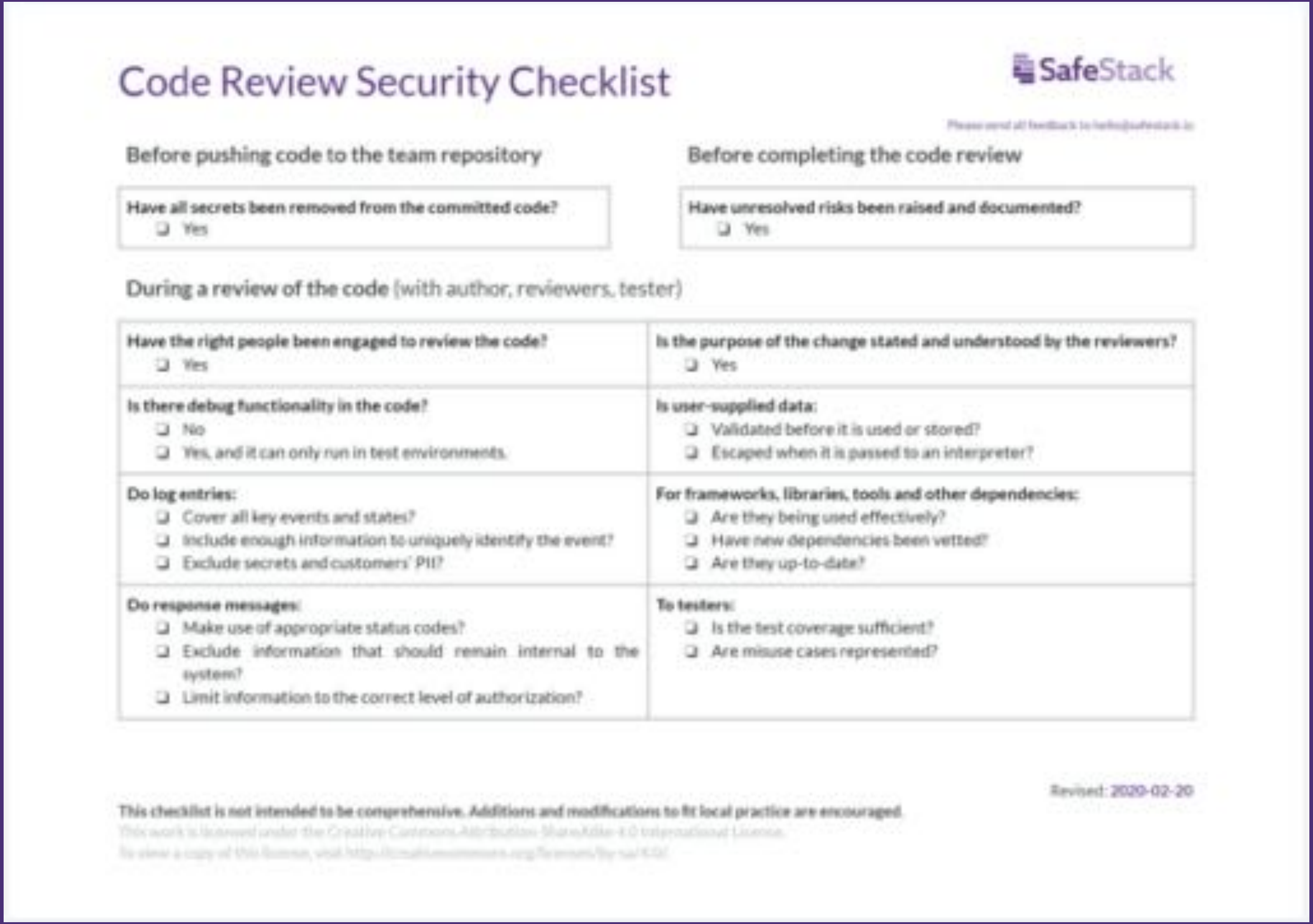
Source:



SafeStack.io

Cynefin framework

Code Review Security Checklist v2



Code Review Security Checklist

SafeStack

Please send all feedback to hello@safestack.io

Before pushing code to the team repository

Have all secrets been removed from the committed code?
- ❑ Yes

Before completing the code review

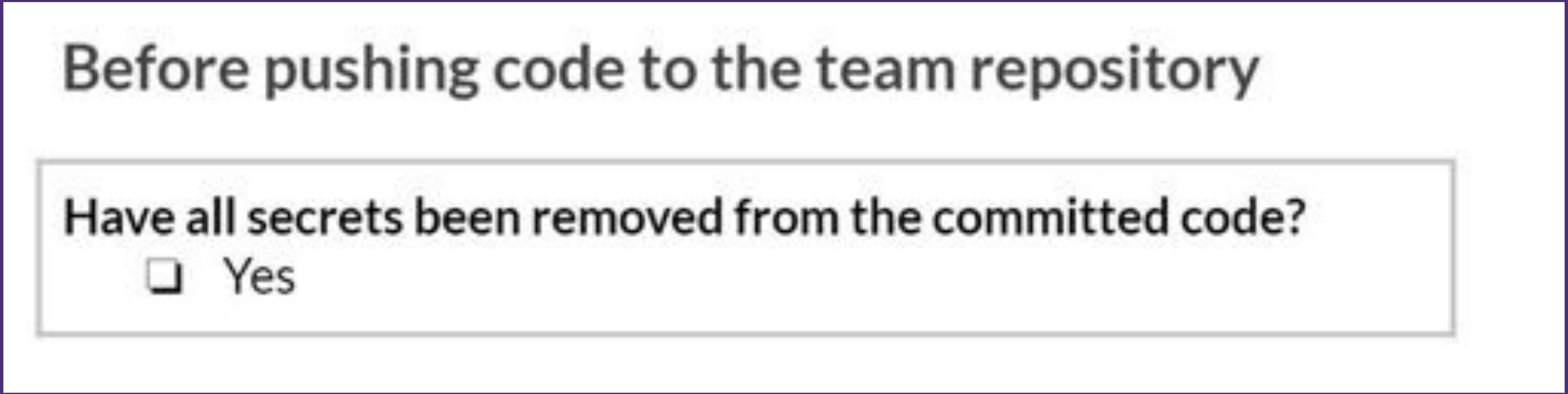
Have unresolved risks been raised and documented?
- ❑ Yes

During a review of the code (with author, reviewers, tester)

Have the right people been engaged to review the code? ❑ Yes	**Is the purpose of the change stated and understood by the reviewers?** ❑ Yes
Is there debug functionality in the code? ❑ No ❑ Yes, and it can only run in test environments.	**Is user-supplied data:** ❑ Validated before it is used or stored? ❑ Escaped when it is passed to an interpreter?
Do log entries: ❑ Cover all key events and states? ❑ Include enough information to uniquely identify the event? ❑ Exclude secrets and customers' PII?	**For frameworks, libraries, tools and other dependencies:** ❑ Are they being used effectively? ❑ Have new dependencies been vetted? ❑ Are they up-to-date?
Do response messages: ❑ Make use of appropriate status codes? ❑ Exclude information that should remain internal to the system? ❑ Limit information to the correct level of authorization?	**To testers:** ❑ Is the test coverage sufficient? ❑ Are misuse cases represented?

Revised: 2020-02-20

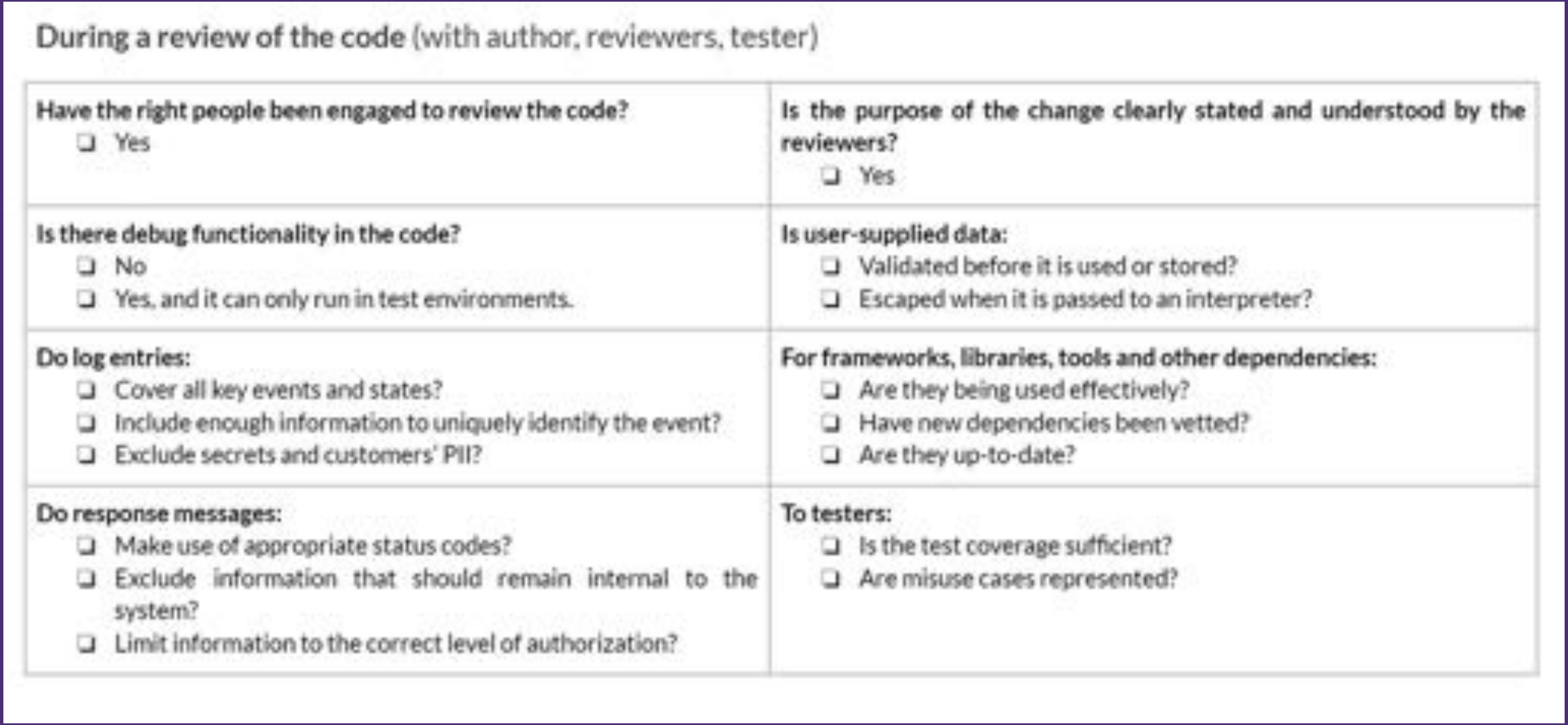
This checklist is not intended to be comprehensive. Additions and modifications to fit local practice are encouraged.



Before pushing code to the team repository

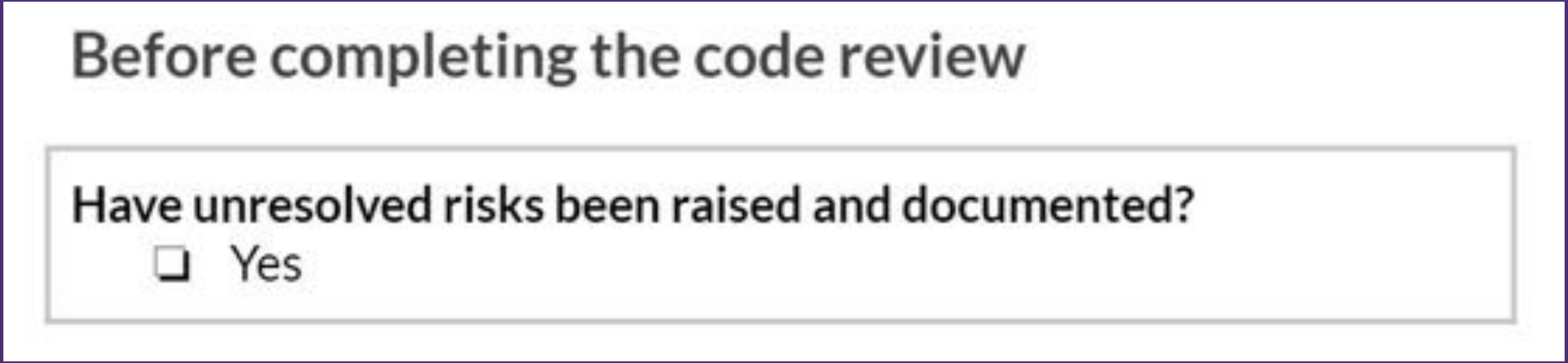
Have all secrets been removed from the committed code?

❑ Yes



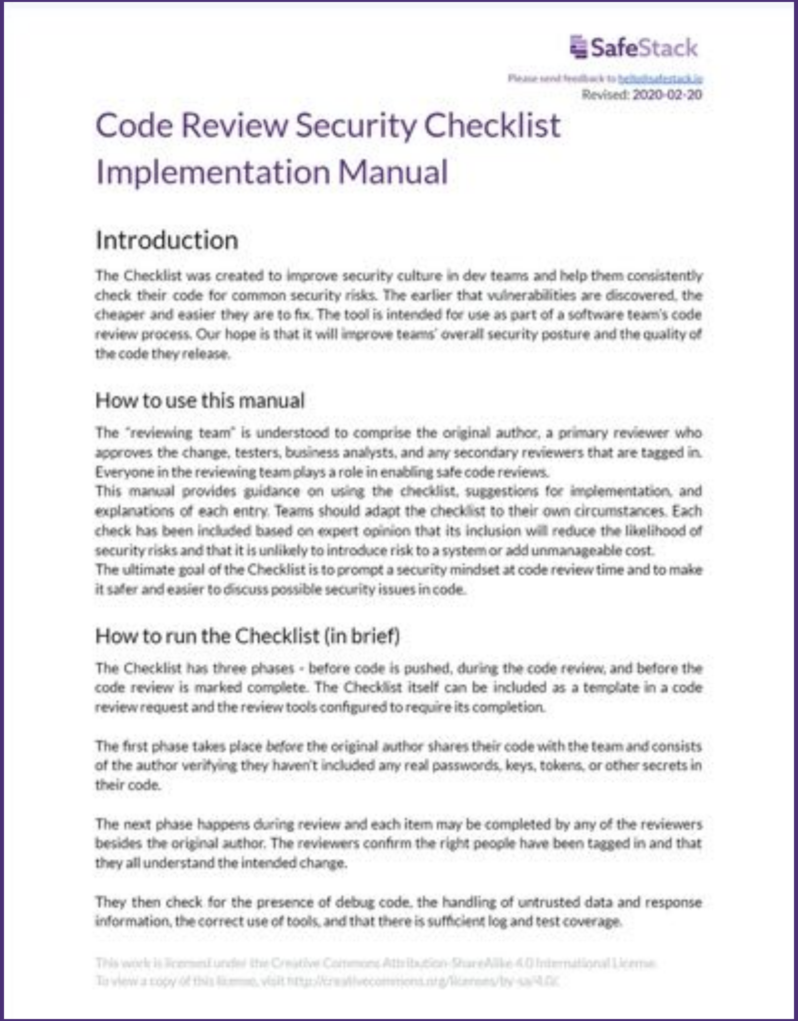
During a review of the code (with author, reviewers, tester)

Have the right people been engaged to review the code? ❑ Yes	**Is the purpose of the change clearly stated and understood by the reviewers?** ❑ Yes
Is there debug functionality in the code? ❑ No ❑ Yes, and it can only run in test environments.	**Is user-supplied data:** ❑ Validated before it is used or stored? ❑ Escaped when it is passed to an interpreter?
Do log entries: ❑ Cover all key events and states? ❑ Include enough information to uniquely identify the event? ❑ Exclude secrets and customers' PII?	**For frameworks, libraries, tools and other dependencies:** ❑ Are they being used effectively? ❑ Have new dependencies been vetted? ❑ Are they up-to-date?
Do response messages: ❑ Make use of appropriate status codes? ❑ Exclude information that should remain internal to the system? ❑ Limit information to the correct level of authorization?	**To testers:** ❑ Is the test coverage sufficient? ❑ Are misuse cases represented?


Before completing the code review

Have unresolved risks been raised and documented?

- ❑ Yes



SafeStack

Please send feedback to hello@safestack.io
Revised: 2020-02-20

Code Review Security Checklist Implementation Manual

Introduction

The Checklist was created to improve security culture in dev teams and help them consistently check their code for common security risks. The earlier that vulnerabilities are discovered, the cheaper and easier they are to fix. The tool is intended for use as part of a software team's code review process. Our hope is that it will improve teams' overall security posture and the quality of the code they release.

How to use this manual

The "reviewing team" is understood to comprise the original author, a primary reviewer who approves the change, testers, business analysts, and any secondary reviewers that are tagged in. Everyone in the reviewing team plays a role in enabling safe code reviews.
This manual provides guidance on using the checklist, suggestions for implementation, and explanations of each entry. Teams should adapt the checklist to their own circumstances. Each check has been included based on expert opinion that its inclusion will reduce the likelihood of security risks and that it is unlikely to introduce risk to a system or add unmanageable cost.
The ultimate goal of the Checklist is to prompt a security mindset at code review time and to make it safer and easier to discuss possible security issues in code.

How to run the Checklist (in brief)

The Checklist has three phases - before code is pushed, during the code review, and before the code review is marked complete. The Checklist itself can be included as a template in a code review request and the review tools configured to require its completion.

The first phase takes place *before* the original author shares their code with the team and consists of the author verifying they haven't included any real passwords, keys, tokens, or other secrets in their code.

The next phase happens during review and each item may be completed by any of the reviewers besides the original author. The reviewers confirm the right people have been tagged in and that they all understand the intended change.

They then check for the presence of debug code, the handling of untrusted data and response information, the correct use of tools, and that there is sufficient log and test coverage.

Your checklist should be

- **Focused** on critical issues not covered by other controls
- **Quick** to run through
- Full of clearly **actionable** items to prevent confusion
- **Shared** with the team collaboratively
- **Tested** in small increments
- **Integrated** into the team’s workflow

Remember the handy acronym: **F.Q.A.S.T.I.**

Introducing the Checklist

- Build a team of enthusiastic people
- Reach across roles and include leaders and managers
- Start small with one team and one system
- Actively seek feedback and incorporate it
- Track improvements

Next Steps

- Establish standard metrics
- Find excited teams to pilot with
- Specialised checklists?
- Lessons from Listo and goSDL?

Application Security Verification Standard

https://github.com/OWASP/ASVS

Top 10 Proactive Controls

https://owasp.org/www-project-proactive-controls/

Sharing is caring

Code Review Security Checklist

https://www.safestack.io/resources-events

Daniel Zollinger
Security Consultant

@dznz
daniel@safestack.io

Let's talk!