

Session 09

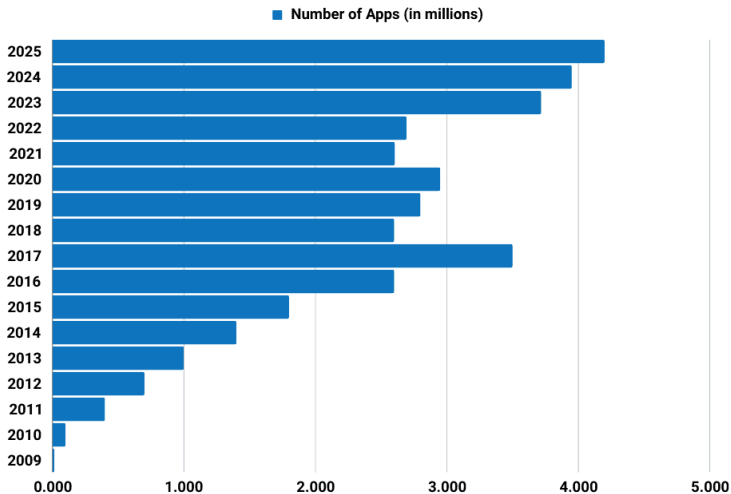
Software Security Assurance

Security of Information Systems (SIS)

Computer Science and Engineering Department

December 10, 2025

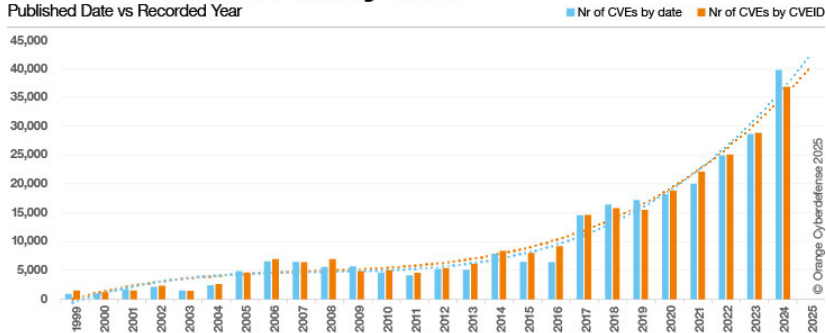
How Many Apps are on Google Play



| Debian Version | Number of Packages |
|----------------|--------------------|
| Debian 1.2 | 848 |
| Debian 8 | ~43k |
| Debian 9 | ~51k |
| Debian 10 | ~59k |
| Debian 11 | ~59.5k |
| Debian 12 | ~64k |
| Debian 13 | ~70k |

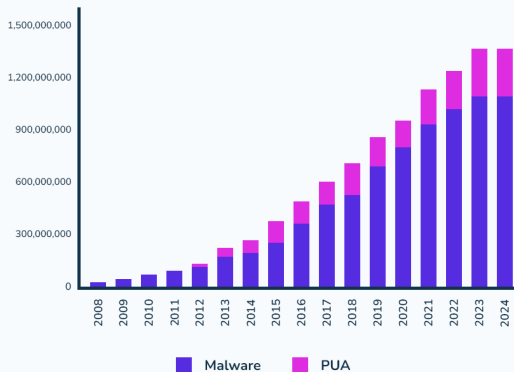
CVEs Published every Year

Published Date vs Recorded Year



<https://thehackernews.com/2025/05/beyond-vulnerability-management-cves.html>

Total Amount of Malware and PUA (by year)



STATIONX

<https://thehackernews.com/2025/05/beyond-vulnerability-management-cves.html>

Papers

- ▶ A survey of static analysis methods for identifying security vulnerabilities in software systems
- ▶ On the capability of static code analysis to detect security vulnerabilities

Problems with Software

- ▶ design and implementation
- ▶ interaction with other components
- ▶ environment / input optimism
- ▶ bugs, vulnerabilities, flaws
- ▶ operational issues: latency, throughput, unresponsiveness

Software Security Assurance

- ▶ ensuring security of software: proper use in potentially harmful environments
- ▶ design analysis
- ▶ secure code review, secure code analysis
- ▶ security testing

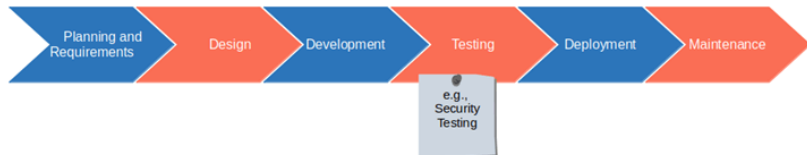
Software Security Framework

| The Software Security Framework (SSF) | | | |
|---------------------------------------|------------------------------|-----------------------|---|
| Governance | Intelligence | SSDL Touchpoints | Deployment |
| Strategy and Metrics | Attack Models | Architecture Analysis | Penetration Testing |
| Compliance and Policy | Security Features and Design | Code Review | Software Environment |
| Training | Standards and Requirements | Security Testing | Configuration Management and Vulnerability Management |

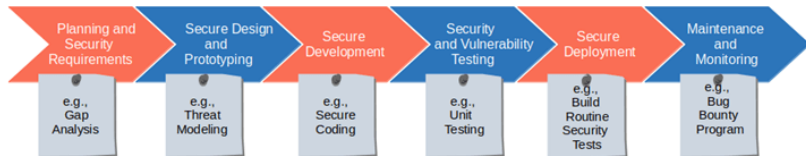
https://www.researchgate.net/publication/239522618_Cyber_Security_Technology_Initiatives

Secure Software Development Life Cycle (SSDLC)

Software Development Life Cycle (SDLC) Process



Secure Software Development Life Cycle (SSDLC) Process



<https://codesigningstore.com/secure-software-development-life-cycle-sdlc>

Software Penetration Testing

- ▶ (ab)using the software in a realistic deployment
- ▶ looking for issues with an attacker mindset
- ▶ ethical hacking, responsible disclosure, reporting of findings

- ▶ Continuous Integration / Continuous Deployment
- ▶ automation of testing, deployment
- ▶ integrate secure testing, secure code analysis
- ▶ ensure security of new features

Ways of Securing Software

- ▶ secure by construction: prevent existence of bugs/vulnerabilities
- ▶ secure environment: prevent exploitation of bugs/vulnerabilities
- ▶ isolated environment: damage control

Secure by Construction

- ▶ providing it as secure (build from specs)
- ▶ building it secure
- ▶ secure before shipping

Secure by Construction (2)

- ▶ formal verification, provably secure
- ▶ programming language features
- ▶ programming practices
- ▶ defensive programming
- ▶ software development process
- ▶ code review
- ▶ code auditing
- ▶ testing
- ▶ fuzzing, symbolic execution

Common Practices/Principles

- ▶ keep it simple: small footprint, few dependencies, no fancy hacks
- ▶ input validation
- ▶ added care when dealing with buffers and strings
- ▶ use linters and static checkers
- ▶ make code readable, document while writing
- ▶ simple and intuitive interfaces
- ▶ mindset: assume the worse
- ▶ do unit tests

Program Analysis

- ▶ focus on applications (i.e. programs) not systems
- ▶ analyze program behavior
- ▶ performance
 - ▶ profiling
 - ▶ reduced resource usage
 - ▶ reduced overhead
- ▶ correctness
 - ▶ debugging
 - ▶ security
 - ▶ robustness
- ▶ no side channel focus

Ways of Doing Program Analysis

- ▶ control flow analysis: reachability
- ▶ data flow analysis: propagation

Types of Program Analysis

- ▶ static analysis: no running of program
- ▶ dynamic analysis: running the program
- ▶ source code analysis: source code is available, use it
- ▶ binary analysis: work on executables and binary files, source code may be unavailable

Static Analysis

- ▶ don't run the program
- ▶ go through its source/binary code
- ▶ control flow and data flow analysis

Dynamic Analysis

- ▶ monitor process
- ▶ usually involves instrumentation
- ▶ valgrind, profilers, Pin
(<https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool>)

Source Code Analysis

- ▶ automated, semi-automated, manual
- ▶ manual: code auditing
- ▶ programming defects, API misuse, lack of compliance, correctness
- ▶ software/code interpretation, pattern matching
- ▶ software formal verification

Binary Analysis

- ▶ reverse engineering
- ▶ binary debugging
- ▶ disassembling, forensics

Terms

- ▶ program comprehension: understand source code
- ▶ code review: fix mistakes, improve code quality and programming practices
- ▶ code auditing: comprehensive analysis with intent of discovering bugs
- ▶ static analysis: automated action performed

Static Analysis

- ▶ analyze computer programs without executing them
- ▶ usually performed on source code
- ▶ automated process

Tools of the Trade

- ▶ editors/reading tools
- ▶ pattern matching tools
- ▶ static analyzers
- ▶ pen & pad

Tools of the Trade (2)

- ▶ open source
 - ▶ Sonar: <http://www.sonarsource.org/> (Java)
 - ▶ Flawfinder: <https://dwheeler.com/flawfinder/> (C/C++)
 - ▶ RATS
 - ▶ Clang Static Analyzer: <http://clang-analyzer.llvm.org/>
 - ▶ Splint: <http://splint.org/> (C) – no longer developed
 - ▶ cppcheck: <http://cppcheck.sourceforge.net/> (C, C++) – plugins for IDEs
- ▶ proprietary
 - ▶ Coverity SAVE: <http://www.coverity.com/products/coverity-save.html>
 - ▶ Klocwork Insight: <http://www.klocwork.com/products/insight/> (C, C++, Java, C#)
 - ▶ CodeSonar: <http://www.grammatech.com/codesonar>
 - ▶ Semmle: <http://semml.com/solutions/>
 - ▶ HP Fortify

Binary Static Analysis

- ▶ requires reverse engineering
- ▶ focused on discovering bugs and creating exploitation PoCs form them to be fixed
- ▶ basic tools: disassemblers, symbol mappers, decompilers
- ▶ automated tools: Veracode, CodeSonar, BitBlaze
- ▶ security analysts, enhancing proprietary solutions

Code Auditing

- ▶ browse source code
- ▶ look for security breaches and possible bugs
- ▶ tools for static code analysis
- ▶ in-depth audit to be done by the developer

Black Box Approach

- ▶ non-open-source code
- ▶ understand protocol or user input format
- ▶ provide “bad” input and test possible violations
- ▶ reverse engineering
- ▶ fuzzing

White Box Approach

- ▶ the “real stuff” – actual code auditing, highlight input processing
- ▶ top-to-bottom: start from main, go down functions
- ▶ bottom-to-top: find all places of external input, system input and start from there

Tools to be Employed

- ▶ static analyzers (cppcheck, Clang Static Analyzer, Coverity)
- ▶ IDA for binary static analysis
- ▶ ctags, cscope, source nav for source code navigation
- ▶ debuggers for runtime analysis
- ▶ valgrind, Rational Purify for dynamic analysis

Code Auditor Requirements

- ▶ know API, OS and machine (background knowledge)
- ▶ recognize patterns (pattern recognition)
- ▶ understand application (functional understanding)
- ▶ audit all code (completeness)

Types of Programs

- ▶ <http://www.ouah.org/mixtercguide.html>
- ▶ setuid/setgid programs
- ▶ daemons and servers
- ▶ frequently run system programs
- ▶ system libraries (libc)
- ▶ widespread protocol libraries (kerberos, ssl)
- ▶ administrative tools
- ▶ CGI scripts, server plugins

Classes of Bugs to Audit

- ▶ API-based bugs
- ▶ external resource interactions
- ▶ programming construct errors
- ▶ state mechanics

API-based Bugs

- ▶ misuse of OS, library or framework APIs
- ▶ dangerous string or formatting functions: e.g., `sprintf()`, `strcpy()`, `strcat()`, `printf()`, `syslog()` . . .
- ▶ dangerous implicit behavior: e.g., allocators that round
- ▶ cumbersome/complicated API reference contents: e.g., threading, IPC

External Resource Interactions

- ▶ privilege escalation through IPCs
- ▶ `system()`, `execve()`, `CreateProcess()`
- ▶ file interaction

Programming Construct Errors

- ▶ CWE: Common Weakness Enumeration
<https://cwe.mitre.org/data/index.html>
- ▶ integer signedness
- ▶ integer boundaries
- ▶ checks that are logically wrong or susceptible to integer problems
- ▶ loops that have bad boundaries
- ▶ unchecked variables

State Mechanics

- ▶ programs left in an inconsistent state
- ▶ thread safety issues
- ▶ async-safety issues
- ▶ global variables left in an undesired state

Methodology

- ▶ target components, meta targeting
- ▶ grep targeting – won't provide understanding
- ▶ read code quickly – ignore what is not important
 - ▶ copy and move data
 - ▶ input/output

List of Issues

- ▶ implementation bugs (miscalculation, check result, not validate input)
- ▶ data types
- ▶ memory corruption

Defensive Programming

- ▶ sh*t happens
- ▶ assume the worst, program accordingly
- ▶ secure programming / secure coding
- ▶ offensive programming
- ▶ formal verification
- ▶ rewrite vs reuse

Secure Coding

- ▶ <https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard>
- ▶ techniques for building secure programs
- ▶ handling input
- ▶ working with memory and buffers
- ▶ handle error/exceptions
- ▶ handling data types

Input Validation

- ▶ anything can be malicious
- ▶ look for injections
- ▶ take into account encoding
- ▶ only allow required format

Buffer Management

- ▶ start address and length
- ▶ boundary checking
- ▶ indexes

String Management

- ▶ length management
- ▶ NUL-byte termination
- ▶ string truncation
- ▶ printable characters

Integer Management

- ▶ conversions (size)
- ▶ overflows
- ▶ signedness

Problems with Software

- ▶ internal to the software
- ▶ complex, interconnected software components
- ▶ due to interaction with other vulnerable components at runtime
- ▶ the build, deployments system itself may be faulty

Software Supply Chain Attack

- ▶ infrastructure could be vulnerable to misconfiguration attacks
- ▶ external (proprietary or open source) libraries / third-party tools may be vulnerable
- ▶ lack of updates of libraries / third-party tools
- ▶ developer / operators who maliciously or inadvertently introduce vulnerabilities

Software Bill of Materials (SBOM)

- ▶ list all dependencies (and versions) required for building and using the software
- ▶ basis for validating security of components
- ▶ allows automation: check vulnerabilities for versions, vulnerability scanning

Reproducible Builds

- ▶ attacker attacks build infrastructure
 - ▶ source code not affected
 - ▶ executable affected
- ▶ ensure the same build across different environments
- ▶ the same executable / output
- ▶ prevents faults introduced by the build system
- ▶ multiple parts reach a consensus of identical builds

Keywords

- ▶ software security assurance
- ▶ software security framework
- ▶ secure software development life cycle
- ▶ secure by design / implementation
- ▶ program analysis
- ▶ static analysis
- ▶ dynamic analysis
- ▶ source code analysis
- ▶ binary analysis
- ▶ code auditing
- ▶ bugs
- ▶ vulnerabilities
- ▶ programming errors
- ▶ CWE (*Common Weakness Enumeration*)
- ▶ defensive programming
- ▶ secure coding
- ▶ software supply chain security
- ▶ software bill of materials
- ▶ reproducible builds

Resources

- ▶ <https://www.amazon.com/Building-Secure-Software-Addison-wesley-Professional/dp/0321774957>
- ▶ <https://www.amazon.com/Secure-Coding-2nd-Software-Engineering/dp/0321822137>
- ▶ <https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard>
- ▶ https://www.owasp.org/index.php/OWASP_Secure_Coding_Practices_-_Quick_Reference_Guide
- ▶ David Binkley: Source Code Analysis: A Road Map
- ▶ <https://cwe.mitre.org/data/index.html>
- ▶ <https://samate.nist.gov/SRD/testsuite.php>
- ▶ <https://circleci.com/blog/secure-software-supply-chain>
- ▶ <https://reproducible-builds.org/>

References

- ▶ <http://pentest.cryptocity.net/code-audits/>
- ▶ <http://software.intel.com/en-us/articles/collection-of-examples-of-64-bit-errors-in-real-programs>
- ▶ <http://www.ouah.org/mixtercguide.html>
- ▶ <http://www.vanheusden.com/linux/audit.html>
- ▶ <http://spinroot.com/static/>
- ▶ <http://spinroot.com/p10/>
- ▶ The Science of Code Auditing, BlackHat EU 2006
- ▶ <https://www.grammatech.com/products/binary-analysis>
- ▶ <http://bitblaze.cs.berkeley.edu/>
- ▶ <https://www.veracode.com/>