GUARDIAN · DOCUMENTATION

How vulnerability management works, and what Guardian does

Part 1 is the theory: the kinds of vulnerability, how a known flaw is found, named, recorded and scored, and how each kind is rated. **Part 2** is Guardian: what it covers today and what it does not, how it decides what is urgent for us, and how a fix goes from a button to a confirmed scan.

Written for a technical reader who is not a security specialist. Every term is explained the first time it appears, and again in the glossary at the end.

Contents

Part 1 · How vulnerabilities work

2. The journey of a known vulnerability
3. The words: vulnerability, weakness, exploit, attack, risk
4. The categories of vulnerability, and how each is found
5. How each category is identified and rated
6. Who gives a vulnerability its name (CVE, GHSA)
7. The databases, and what each one answers
8. Attacks: how a flaw is used, and the attack vectors
9. The score: CVSS, explained metric by metric
10. How likely is an attack: EPSS, KEV, SSVC
11. From score to risk rating
12. The web entrance: TLS, certificates, headers

Part 2 · What Guardian does

14. Guardian's offering on one page
15. How Guardian rates what it finds, for us
16. Is it really exploitable here? The assessment
17. Guardian from end to end

To look things up

19. Glossary and acronyms
20. References

Version 1 · November 2026 · Guardian is the owner's private vulnerability console. This document contains no data of any application: only how things work.

PART 1

How vulnerabilities work

The theory, for every kind of system: the families of vulnerability, how a known flaw travels from discovery to a fix, how it is named, recorded, scored, and how urgent it really is.

1 · The journey of a known vulnerability

This is the path of a **known vulnerability in a component**: a library, an operating system, a server product. It is the best organised part of the security world, with public names, scores and databases. The other categories (flaws in our own code, secrets, misconfigurations) do not follow this path: chapter 4 explains how they are identified and rated.

1
Found
A researcher, the author or an attacker finds a flaw in a library or product.

2
Reported
It is reported, ideally privately, to the author, who prepares a fix.

3
Named
A CVE number is given (e.g. CVE-2024-12345), and often a GitHub advisory (GHSA).

4
Described
Databases add the details: which versions, which fix, how serious (CVSS), what kind of mistake (CWE).

5
Watched
FIRST forecasts how likely it is to be attacked (EPSS); CISA lists it if attackers already use it (KEV).

6
Matched
A scanner lists the components a system contains, with their exact versions, and checks them against the databases.

7
Weighed
The owner of the system decides how urgent it is in their context: is it reachable, is it being exploited?

8
Fixed
The component is updated or patched; the next scan confirms the flaw is gone.

The one idea to keep. The *existence* of a vulnerability is a fact that comes from outside (the advisory says: "version 1.2.3 of this library has this flaw"). Its *importance for a given system* is a judgement that depends on how that system uses the component. A good process never doubts the fact; it only adjusts the urgency, and only with evidence.

2 · The words

Security people use a handful of words very precisely. Once these are clear, everything else follows.

Word	What it means, in plain terms	Example
Weakness	A *kind* of mistake that programmers make. It is a category, not a specific bug. Catalogued as **CWE**.	"Not checking user input before putting it in a web page" (CWE-79, cross-site scripting).
Vulnerability	A *specific* weakness in a specific product and version, that someone could use to do harm. Named with a **CVE** number.	"Library X up to version 2.4.0 puts user text in pages without escaping it."
Exploit	The concrete method or code that uses a vulnerability.	A crafted link that makes a victim's browser run the attacker's script.
Attack	Someone actually using an exploit against a real system.	The link is sent to thousands of users.
Threat	Who could attack and why (criminals, automated bots, insiders).	Bots that scan the whole internet for one known flaw.
Impact	What is lost if the attack works. Always one of three: **confidentiality** (data is read), **integrity** (data or behaviour is changed), **availability** (the service stops). Together called the **CIA triad**.	Session cookies stolen = confidentiality lost.
Risk	How worried we should be: **how likely** an attack is, times **how bad** the impact would be, **in our situation**.	A serious flaw in code we never run is a low risk.
Advisory	The official notice that describes a vulnerability: affected versions, fixed version, severity.	A GitHub Security Advisory (GHSA).
Patch / fixed version	The version of the software in which the author removed the flaw.	"Fixed in 2.4.1".
Zero-day	A vulnerability attackers use before a fix exists (the author has had "zero days" to fix it).	—
Dependency	A library our application uses. **Direct**: we chose it. **Transitive**: it came in because another library needs it.	We use a PDF library; it uses a compression library.
SBOM	Software Bill of Materials: the full list of components inside an application, with versions. Guardian builds one at every scan.	"385 libraries, 292 in the Android app".

Why is something a vulnerability? Three things must be true together: (1) there is a **flaw** in the code, (2) an attacker can **reach** it, by sending data to it or by being where it runs, and (3) using it causes an **impact** on confidentiality, integrity or availability. If one of the three is missing for us, the vulnerability still exists in the library, but it cannot hurt us. That is exactly what the exploitability assessment (chapter 14) checks.

3 · The categories of vulnerability, and how each is found

Vulnerabilities come in a handful of families. Each family is found in a different place and with a different tool, and is fixed in a different way. Each tile says what the family is and gives examples; the next page says where each is found and with which tool. What Guardian covers is in Part 2.

1 Vulnerable components

Software written by others, with a known CVE. Fixed by updating or patching.

Application: Spring, Python or npm libraries. Infrastructure: OS patches (Windows KBs, Linux packages), agents, database engines, container images.

2 Flaws in our own code

Mistakes in how the application is written, so an attacker's input is trusted or a check is missing.

SQL injection, XSS, missing access control, weak login, SSRF, unsafe deserialization.

3 Exposed secrets

Keys, passwords and tokens written where others can read them.

An API key committed to Git, a password in a log or in a container image.

4 Misconfiguration and drift

Settings that differ from a safe baseline (usually the CIS Benchmarks), in the cloud or on servers on-premises.

An open port, ACLs or file permissions too wide, a public bucket, default passwords, cloud roles with too much power.

5 External exposure

Everything that can be reached from the internet, and how well its entrance is protected.

TLS versions and cipher suites, certificates, security headers, admin pages left public, forgotten subdomains.

6 Supply chain

The way software reaches us: packages and build tools that are malicious or tampered with.

A look-alike package published to steal data, a compromised build step. OWASP Top 10:2025, A03.

7 Design and logic flaws

The design itself lacks a protection; each piece works, but together they can be abused.

No limit on password attempts, a payment step that can be skipped, a reset link that never expires.

Penetration testing is a method, not a category. A penetration tester attacks the application for real, with permission, and can find problems of every family above. It is the only way to find most design and logic flaws (7), because no scanner knows what the application is supposed to do.

3 · Where each family is found, and with which tool

Family	Where it is found	Tools (and what the letters mean)
1 Components, application	The repository: lockfiles, build files, the libraries shipped	**SCA**, Software Composition Analysis
1 Components, infrastructure	The machines' file systems: OS packages, KBs, installed agents, middleware, database engines; container images	**Vulnerability scanners** such as Tenable Nessus or Qualys VMDR; container image scanners
2 Our code	The source code; the running application	**SAST**, Static Application Security Testing (reads the code); **DAST**, Dynamic Application Security Testing (attacks the running app); penetration test
3 Secrets	The repository and its whole history; logs; images	Secret scanners
4 Misconfiguration	Cloud accounts; servers and network devices on-premises	**CSPM**, Cloud Security Posture Management; configuration checks against the **CIS** Benchmarks
5 External exposure	The public addresses, as seen from the internet	**EASM**, External Attack Surface Management; TLS testers (testssl.sh); header checks
6 Supply chain	Package registries, build pipelines	SCA with malware advisories; signed builds
7 Design and logic	How the application behaves	Penetration test; threat modelling (thinking through how the design could be abused)

Where do attacks on a database belong?

Family 2 · code SQL injection: the application builds a query from what the user typed, so the user can rewrite it.

Family 4 · configuration a database reachable from the internet, a default password, a user with every right.

Family 1 · component a CVE in the database engine itself (PostgreSQL, MySQL…): fixed by updating it.

The same goes for many attacks: the name says *what* happens, the family says *where the mistake is*, and so who fixes it and how.

4 · How each category is identified and rated

Only known flaws in components have a CVE and a CVSS score. The other categories have their own way of being named and rated. This table puts them side by side.

Category	How a finding is identified	Who rates it, and how	Has a CVE?
1 Components	CVE, GHSA or other OSV id; the package and its exact version	CVSS, given by the CNA, NIST's NVD or GitHub (chapter 8); then EPSS and KEV for how likely an attack is (chapter 9)	Yes
2 Our code	The weakness type (CWE) and the scanner's rule; file and line	The scanning tool's own severity (Semgrep: error, warning, info). A penetration tester usually scores with CVSS or with the OWASP Risk Rating Methodology (likelihood × impact)	No: it is our own code, not a published product
3 Secrets	The kind of key (the scanner's rule), file, line and commit	Treated as high by default: a valid key gives direct access, no exploit is needed. Removing the line is not enough, because Git keeps the history: the key must be revoked and replaced	No
4 Misconfiguration	The control of a safe baseline that is not met, e.g. a CIS Benchmark recommendation, or a cloud tool's rule	The baseline's level (CIS Level 1, the essential settings; Level 2, extra protection) and the tool's severity, raised when the system faces the internet	No
5 External exposure	The check's name; known TLS attacks have a CVE (Heartbleed CVE-2014-0160, POODLE CVE-2014-3566)	CVSS for the named attacks; otherwise the testing tool's severity, following OWASP and MDN guidance	Sometimes
6 Supply chain	A malware advisory (MAL-… or a GitHub malware advisory)	Always critical: the package was published to do harm. It is removed, not updated	Usually not
7 Design and logic	A penetration-test finding, with its CWE	The penetration tester, with CVSS or the OWASP Risk Rating Methodology	No

In short. CVE, CVSS, EPSS and KEV belong to the first category, the only one shared by the whole world (everybody uses the same libraries). Everything that is ours alone (our code, our keys, our settings) is named by the tool that finds it and rated by its severity and by our context.

5 · Who gives a vulnerability its name

CVE · Common Vulnerabilities and Exposures

This chapter is about category 1, known flaws in components. CVE is the worldwide catalogue of publicly known vulnerabilities. Each one gets a unique number in the form `CVE-YEAR-NUMBER`, for example `CVE-2021-44228` (the famous "Log4Shell"). The number lets everyone (vendors, scanners, governments) talk about exactly the same problem.

- **Who runs it:** the CVE Program, operated by MITRE (a US non-profit) with sponsorship from CISA, the US cyber-security agency. MITRE and CISA are the two top-level "roots" of the program.
- **Who assigns the numbers:** hundreds of **CNAs** (CVE Numbering Authorities): software companies (Google, Microsoft, GitHub…), security teams and national CERTs. Each one numbers the flaws in its own area. When no CNA covers a product, MITRE acts as "CNA of last resort".
- **Life of a CVE record: Reserved** (the number is booked while the fix is prepared, details still secret) → **Published** (at least a description and one public reference) → possibly **Rejected** (the number should not be used, e.g. a duplicate; it stays visible as invalid).

GHSA · GitHub Security Advisories

GitHub runs its own advisory database for open-source packages. Its IDs look like `GHSA-xxxx-xxxx-xxxx`. Most advisories also carry a CVE number; some only have a GHSA. GitHub sorts them into **reviewed** advisories (checked by GitHub's team and linked to exact package versions), **unreviewed** ones (imported automatically, e.g. from NVD) and **malware** advisories (packages published on purpose to do harm). It uses four severity levels: LOW MODERATE HIGH CRITICAL, the same Guardian shows.

> **Why two names?** CVE says *what* the problem is. GHSA (and OSV, next chapter) adds the precise *package names and version ranges* a program needs to match a vulnerability against a list of libraries automatically. Guardian shows the GHSA (or other OSV id) and uses the CVE number to look up KEV and EPSS.

Other names you may meet

Prefix	From
`CWE-79`	Not a vulnerability: a weakness *type* (chapter 6).
`PYSEC-…`, `GO-…`, `RUSTSEC-…`	Advisories of the Python, Go and Rust ecosystems, published through OSV.
`MAL-…`	Malicious packages, collected by the OpenSSF.

6 · The databases, and what each one answers

No single database knows everything. Each one answers a different question; Guardian combines them.

Source	Run by	The question it answers	Used by Guardian
CVE List (cve.org)	CVE Program (MITRE, CISA, CNAs)	Does this vulnerability exist, and what is its official name?	Through the CVE ids in each advisory
NVD (nvd.nist.gov)	NIST, US government	How serious is it (CVSS), what kind of mistake (CWE), which products (CPE names)?	Links for CVEs found by the web scan
GitHub Advisory Database	GitHub	Which versions of which open-source package are affected, and which version fixes it?	Yes, through OSV.dev
OSV.dev	Google and the OpenSSF	Given a package and its exact version, which known vulnerabilities apply?	**Yes: the core of the library scan**
CWE (cwe.mitre.org)	MITRE	What type of mistake is this, how does it happen, how is it prevented?	Links on code findings
EPSS (first.org)	FIRST	How likely is it to be exploited in the next 30 days?	**Yes: a guardrail**
KEV (cisa.gov)	CISA, US government	Are attackers already using it?	**Yes: a guardrail**

OSV.dev, the one Guardian relies on most

OSV ("Open Source Vulnerabilities") is an aggregator: it collects the advisories of many ecosystems (GitHub Security Advisories, PyPA for Python, RustSec, Go and others) in one common format, the OpenSSF OSV schema. Its strength is precision about versions: each advisory lists **ranges** made of events. An `introduced` event says "from this version the flaw is present", a `fixed` event says "from this version it is gone". So when Guardian asks "is `jspdf 2.5.1` affected?", the answer is exact, not a guess based on product names.

CWE, the catalogue of mistakes

MITRE defines CWE as "a community-developed list of common software and hardware weaknesses", where a weakness is "a condition in a software, firmware, hardware, or service component that, under certain circumstances, could contribute to the introduction of vulnerabilities". The list is sponsored by CISA and updated three to four times a year. A CWE is a *type* of flaw; a CVE is one *instance* of it in a product. One CWE is behind thousands of CVEs. Examples: **CWE-79** cross-site scripting (user input put in a page unescaped), **CWE-89** SQL injection, **CWE-22** path traversal (reading files outside the allowed folder), **CWE-200** exposure of sensitive information. MITRE publishes every year the "CWE Top 25 Most Dangerous Software Weaknesses".

7 · Attacks: how a flaw is used

A vulnerability is only dangerous if an attacker can deliver something to it. The path the attacker takes is the **attack vector**. CVSS (next chapter) uses four of them, from the most to the least exposed:

Vector	Where the attacker must be	Example
Network (AV:N)	Anywhere on the internet. The most dangerous: anyone, anywhere, also automated bots.	Sending a crafted request to a web site.
Adjacent (AV:A)	On the same local network (same Wi-Fi, same office network, same Bluetooth range).	A flaw in a router's local interface.
Local (AV:L)	Already on the machine, with an account or by convincing a user to open a file.	A malicious file opened on the computer.
Physical (AV:P)	Holding the device in their hands.	Plugging something into a USB port.

The most common families of attack on web applications

OWASP, the open community for web application security, publishes the **OWASP Top 10**, the ten most important risk categories for web applications. The current edition, **2025**, is:

#	Category	In plain words
A01	Broken Access Control	Users can see or do what they should not (another user's data, admin pages). Since 2025 it also includes server-side request forgery.
A02	Security Misconfiguration	Unsafe settings: debug pages on, default passwords, missing security headers.
A03	Software Supply Chain Failures	Vulnerable or malicious libraries and build tools. **This is what Guardian's library scan covers.**
A04	Cryptographic Failures	Data not encrypted, or encrypted badly (old TLS, weak algorithms).
A05	Injection	Attacker text treated as code: SQL injection, cross-site scripting, command injection.
A06	Insecure Design	The design itself has no protection (e.g. no limit on password attempts).
A07	Authentication Failures	Weak login, sessions that can be stolen or never expire.
A08	Software or Data Integrity Failures	Code or data accepted without checking it was not changed (unsigned updates, scripts without integrity).
A09	Security Logging and Alerting Failures	Attacks happen and nobody notices.
A10	Mishandling of Exceptional Conditions	Errors handled badly: crashes, error pages that reveal internals, failing "open" instead of "closed".

The 2021 edition, still often quoted, had Injection third and Vulnerable and Outdated Components sixth.

8 · The score: CVSS

CVSS (Common Vulnerability Scoring System) is the standard way to say *how severe* a vulnerability is, with a number from **0.0 to 10.0**. It is maintained by FIRST, the global forum of incident response and security teams. Whoever publishes the advisory (the CNA, GitHub, NIST's NVD) usually calculates it.

> **Severity is not risk.** CVSS describes the vulnerability in general, for the worst reasonable case. It does not know how *we* use the software, and it does not say how likely an attack is. FIRST itself says CVSS should not be used alone to assess risk. That is why Guardian adds EPSS, KEV and its own assessment.

From number to word

Score	Rating	As Guardian shows it
0.0	None	—
0.1 – 3.9	Low	LOW stored, not counted
4.0 – 6.9	Medium	MODERATE
7.0 – 8.9	High	HIGH
9.0 – 10.0	Critical	CRITICAL

How the number is made: the base metrics (CVSS 3.1)

The score is not chosen by feeling. The analyst answers eight questions; a published formula turns the answers into the number. The answers are written as a short code, the **vector**.

Metric	The question	Possible answers (most dangerous first)
AV Attack Vector	From where can the attacker strike?	N Network · A Adjacent · L Local · P Physical
AC Attack Complexity	Does the attack need special conditions outside the attacker's control?	L Low (works every time) · H High
PR Privileges Required	Does the attacker need an account first?	N None · L Low (normal user) · H High (administrator)
UI User Interaction	Must a victim do something (click, open a file)?	N None · R Required
S Scope	Can the damage spread beyond the vulnerable component (e.g. escape a sandbox)?	C Changed · U Unchanged
C Confidentiality	How much data can be read?	H High · L Low · N None
I Integrity	How much can be changed?	H High · L Low · N None
A Availability	Can the service be stopped?	H High · L Low · N None

Reading a vector

```
CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
```

Read it left to right: version 3.1; attackable **from the internet**; **easy**; **no account** needed; **nobody has to click**; stays in the component; **everything** can be read, changed and stopped. This is the worst combination and

scores **9.8 Critical**. Change only `AV:N` to `AV:L` (the attacker must already be on the machine) and the same flaw drops to 8.4 High: the place the attacker must be in changes everything.

The other groups, and CVSS 4.0

CVSS 3.1 also has **Temporal** metrics (is there exploit code, is there a fix) and **Environmental** metrics (the same questions answered for one specific organisation). In practice advisories publish only the Base score.

CVSS 4.0 (2023) has four metric groups (Base, Threat, Environmental, Supplemental) and changes some parts: the new **AT, Attack Requirements** metric is split from Attack Complexity (conditions in the target, such as a particular setting, versus the attacker's effort); **Scope** is removed and impact is split between the vulnerable system and other systems; Temporal becomes **Threat**, with only **Exploit Maturity** (Attacked, Proof-of-concept, Unreported; when nobody fills it in it counts as Attacked, the worst case); an optional **Supplemental** group (Safety, Automatable, Recovery, Value Density, Response Effort, Provider Urgency) adds context without changing the number. Scores are labelled by what they include: **CVSS-B** (base), **CVSS-BT** (base and threat), **CVSS-BE** (base and environmental), **CVSS-BTE** (all three).

9 · How likely is an attack: EPSS, KEV, SSVC

CVSS answers "how bad could it be?". Tens of thousands of CVEs are published every year (about 40,000 in 2024), and most are never used by any attacker. Two free public sources answer the other question, "will it actually be used?".

EPSS · Exploit Prediction Scoring System

- **What it is:** in FIRST's words, "a data-driven machine-learning model that estimates the probability that a published CVE will be exploited in the wild in the next 30 days". It gives every CVE a **probability from 0 to 1 (0% to 100%)**.
- **How it is made:** from real signals: whether exploit code is public, mentions in security lists and social media, the kind of flaw, its age, and attack activity seen by sensors around the world.
- **Updated every day**, so a score can rise quickly when an exploit appears. Free API: `api.first.org/data/v1/epss`.
- **The percentile** says how a CVE ranks against all others: "95th percentile" = more likely than 95% of all CVEs. Exploitation is rare, so most scores are tiny: 10% is already a very high score.

In Guardian: EPSS of **10% or more** is a guardrail: the rating is never lowered, whatever our code does.

KEV · Known Exploited Vulnerabilities catalog

- **What it is:** the list kept by **CISA** (the US Cybersecurity and Infrastructure Security Agency) of vulnerabilities that **attackers are already using in real attacks**. Not a forecast: a fact.
- **To enter it a vulnerability must:** (1) have a CVE number, (2) have reliable evidence of active exploitation, (3) have a clear remediation, such as an update.
- **Why it exists:** Binding Operational Directive **BOD 22-01** (2021) obliges US federal agencies to fix every KEV entry within set deadlines. Everyone else is encouraged to treat KEV as the top priority.

In Guardian: a CVE in KEV is **never** rated lower. Even if our usage makes it hard to reach, someone out there is attacking it right now, with tools that improve.

SSVC · Stakeholder-Specific Vulnerability Categorization

A decision tree used by CISA: instead of a number it answers with an action, from the state of exploitation, the technical impact, whether the attack can be automated, how important the system is and the impact on public well-being. The four outcomes are **Track** (no action needed now, keep watching), **Track*** (watch more closely), **Attend** (deal with it soon, people above are informed) and **Act** (deal with it as soon as possible). Guardian's guardrails follow the same spirit: exploitation in the wild weighs more than the theoretical score.

10 · From score to risk rating

A **severity** (like CVSS) describes a flaw in general. A **risk rating** answers "how urgent is this *for this system*?". It combines three things:

How bad
The severity: what an attacker could do (CVSS, or the tool's severity).

How likely
Is it being exploited (KEV)? Will it be soon (EPSS)? Can an attacker reach it here?

How much it matters here
Is the system on the internet? Does it hold personal data? Is the vulnerable part even used?

The result is usually written in the same four words (low, moderate, high, critical), but it is a different thing from the severity: two companies with the same library can rightly give it two different risk ratings. Whoever lowers a rating writes down why; the standard way to write it is VEX.

VEX: the standard way to say "not affected"

VEX (Vulnerability Exploitability eXchange), promoted by CISA, is a standard statement that says, for a product and a vulnerability, one of four statuses: **Not affected**, **Affected**, **Fixed**, **Under investigation**. "Not affected" must give one of five justifications:

Justification	Meaning
`component_not_present`	The vulnerable component is not in the product at all.
`vulnerable_code_not_present`	The component is there, but not the vulnerable part of it.
`vulnerable_code_not_in_execute_path`	The vulnerable code is there but is never run.
`vulnerable_code_cannot_be_controlled_by_adversary`	It runs, but an attacker cannot send it the input needed.
`inline_mitigations_already_exist`	Something else already blocks the attack.

A VEX statement is how a software maker tells its customers "this CVE is in our product, but it cannot hurt you, because…". Guardian uses the same justifications for its own ratings (Part 2).

11 · The web entrance: TLS, certificates, headers

Every public web address is an entrance. Its protection is checked from outside, the way a browser sees it, plus the questions a security tester asks.

Encryption in transit: TLS

- **TLS** (Transport Layer Security) is the encryption behind `https://`. Versions: SSL 2 and 3 (broken), TLS 1.0 and 1.1 (retired, weak), **TLS 1.2** and **TLS 1.3** (current).
- A **cipher suite** is the set of algorithms used in one connection (key exchange, encryption, integrity). Old ones (RC4, 3DES, export ciphers) are weak.
- **Forward secrecy**: even if the server's key is stolen later, past traffic stays secret.
- Known attacks on old TLS have names: **Heartbleed** (memory leak in OpenSSL), **POODLE** (SSL 3), **BEAST** and **Lucky13** (old CBC ciphers), **SWEET32** (64-bit block ciphers), **FREAK**, **Logjam** and **DROWN** (weak or export keys), **ROBOT** (RSA key exchange), **CRIME** and **BREACH** (compression).
- The **certificate** proves the site is who it says: it must be valid, not expired, for the right name, from a trusted authority. A **CAA** DNS record lists which authorities may issue certificates for a domain.

Security headers and cookies

Header	What it protects against
Strict-Transport-Security (HSTS)	Being downgraded to plain http by an attacker on the network.
Content-Security-Policy (CSP)	Injected scripts (cross-site scripting): only the listed sources may run.
X-Frame-Options / frame-ancestors	Clickjacking: the site shown invisibly inside another site.
X-Content-Type-Options	The browser guessing a file's type and running it as script.
Referrer-Policy, Permissions-Policy, Cross-Origin-Opener-Policy	Leaking addresses to other sites; unwanted access to camera, microphone, location; other windows keeping a handle on ours.
Cookie flags: Secure, HttpOnly, SameSite	Cookies sent over plain http, read by scripts, or sent with requests from other sites.

A web scan also looks for files that must never be public (`.git`, `.env`, backups, source maps), dangerous CORS settings, the TRACE method, error pages that show internals, and, for our own domains, the e-mail protections **SPF** (who may send mail for the domain) and **DMARC** (what to do with forged mail).

PART 2

What Guardian does

What Guardian covers today and what it does not, how it rates what it finds for our applications, and how a fix goes from a button in the console to a confirmed scan.

12 · Guardian's offering on one page

Guardian is the owner's private vulnerability console. It watches four applications today: HINT 365 (an Android app and its web app), Tandem AI, Una Zampa Avanti and CoachingPlatform (web apps). The tiles say what it covers, using the categories of chapter 3.

Covered · Libraries (SCA)

Every library each app ships: the full Android tree for the phone app, the npm, pnpm and bundled libraries for the web apps, checked against OSV.dev. Each advisory gets an exploitability assessment (chapter 14).

Category 1, application side · and category 6, malicious packages

Covered · Our code (SAST)

The applications' own code, read by Semgrep with its open-source rules: injection, cross-site scripting, unsafe functions.

Category 2, static part

Covered · Secrets

Keys and passwords in each repository and in its whole Git history, found by gitleaks. The value is never stored, only rule, file and line.

Category 3

Covered · Web addresses

The public addresses: TLS and cipher suites, certificate, security headers, cookies, CORS, exposed files, error pages, e-mail protections of our own domains.

Category 5

Planned · Penetration test

Real, authorised attacks on the running web apps with test accounts (DAST and a tester's work), to find what scanners miss.

Category 2, dynamic part · category 7

Not covered · Configuration and drift

Cloud settings (Supabase row-level security, R2 buckets, Cloudflare and Vercel settings), open ports, ACLs and permissions. A possible next step for our own cloud settings.

Category 4

Not covered · Infrastructure

Operating systems, OS patches (KBs), installed agents, database engines, container images. Not needed today: our apps run on Cloudflare, Vercel and Supabase, which patch their own machines.

Category 1, infrastructure side

Around the scans

A Fix button that opens a task for the developer, a daily routine that works on it, a scan that confirms it; this documentation; every false positive written down with its reason.

For every covered category

How many vulnerabilities do we have? The numbers change at every scan, so they are not in this document: they are in the console, on the **Report** tab (open, critical, high, in remediation, fixed in 30 days; by application and by risk), one tab per family (application vulnerabilities, web URL scanning). Every number is counted from the very list it opens.

13 · How Guardian rates what it finds, for us

Guardian starts from the severity the advisory or the tool gives (chapter 4), and adjusts it to our situation, always with a written reason. This is Guardian's risk rating (chapter 10).

Situation	What Guardian does
The library is used only to **build** the app (compilers, test tools, the development server) and never reaches the users	Stored, rated one level lower, **not counted or shown**. A flaw in a tool that never leaves the developer's machine or the build server cannot reach a user.
A flaw in the **hosting platform's** encryption settings (Vercel, Cloudflare workers.dev)	Stored as `platform`, not counted: the platform decides those settings, we cannot change them.
A **false positive** (the scanner was wrong)	Recorded with its reason in the decisions file or the scanner's README, never silently hidden.
A **low** rating	Stored, not counted in the totals (the owner's choice), still visible in the scan's results.
A **runtime** library with a known flaw	The **exploitability assessment** (next chapter) decides whether it stays at the advisory's level or is lowered, with guardrails.
The owner's **written decision** (accepted risk, false positive)	Always wins, and is shown as the owner's decision.

Low ratings, build tools, platform settings and recorded false positives are kept in the database but left out of every count and list in the console, so that what is shown is what needs work.

14 · Is it really exploitable here? The assessment

This applies to category 1, the library advisories, the only ones with CVE, CVSS, EPSS and KEV to weigh. For every library advisory, at every scan, Guardian writes a short report: what it checked, what it found, the verdict and why. The verdict is either **CONFIRMED** (the advisory's level stays) or **LOWERED** (by one or two levels). The rules are designed so that missing information never helps a downgrade.

Step 1 · Guardrails: if any is true, the rating is confirmed

1. The CVE is in **CISA KEV**: attackers already use it.
2. **EPSS ≥ 10%**: an attack in the next 30 days is likely.
3. The advisory has **no CVE**, or KEV or EPSS **could not be read** that day: we could not check the first two rules.
4. The vector says **internet + no account + no click** (AV:N/PR:N/UI:N) **and our own code imports the library**: the classic recipe of a mass attack.

Step 2 · Evidence, in three independent families

Family	What can count towards lowering	What does *not* count
How we use it	The library is declared but no file of ours imports it; or our code imports it but never calls the vulnerable function named by the advisory.	"We don't import it" when another library pulls it in: its code can still run through that library.
What the attack needs (CVSS)	The attacker must be on the machine, on the local network or holding the device (AV:L, AV:A, AV:P); needs administrator access (PR:H); a victim must act on a non-internet attack (UI:R).	Internet attacks; a normal account.
Where it can run	The advisory concerns only Windows or macOS, and we run on Linux servers, browsers and Android.	Anything not clearly limited.

Step 3 · Counter-check: try to prove the evidence wrong

Each piece of evidence that would lower the rating is tested again, more widely. If the library's name appears anywhere in our code (it might be loaded in an unusual way), "not imported" is dropped. If a name like the vulnerable function appears anywhere, "never called" is dropped. If the advisory also mentions Java, "Windows only" is not trusted for an Android app.

Step 4 · Verdict

- Evidence from **two** families → **one level lower**. From **three** → **two levels lower**. Never more.
- Never below LOW; never "not affected".
- One family alone is not enough: a single argument can be wrong.
- Everything is redone at every scan: if the evidence changes (new EPSS, the library now imported), the verdict changes too.
- A lowered rating is still a vulnerability: the library still has to be updated. It is only less urgent.

Example. A HIGH path-traversal flaw affecting only Windows, attackable only locally, in a library pulled in by another one. Guardrails: not in KEV, EPSS 0.1%, it has a CVE. Evidence: *how we use it* neutral (it comes through another library); *what the attack needs* lowers (local only); *where it runs* lowers (Windows only, we run on Linux). Two families → **LOWERED from HIGH to MODERATE**. If the next day EPSS rose to 12%, it would be **CONFIRMED at HIGH** again.

15 · Guardian from end to end

1. **Census.** `apps.json` lists the applications watched and their parts: **app** (Android / iOS) and **web app** (web pages and their engine).
2. **Scan.** Every second Monday night (Tandem AI every Monday), GitHub Actions checks each application out read-only and runs: the library inventory (the full Android tree for the app, the shipped npm or pnpm libraries for the web), OSV.dev, the exploitability assessment, Semgrep on our code, gitleaks on the whole history. Then the web URL scan reads the public addresses.
3. **Store.** Results go into Guardian's database (Cloudflare D1, in the EU): only codes, library names, file paths, addresses and counts. Never any user's data. An incomplete scan never replaces the last complete one.
4. **Show.** The console, behind Cloudflare Access (only the owner): totals, charts, one box per application, every finding with its explanation and the assessment. Every number is counted from the very list it opens.
5. **Fix.** The owner presses Fix: Guardian opens a task (an issue labelled `guardian-fix`) in the application's repository, with the finding and the assessment. A daily routine picks it up and prepares the change following that repository's own rules.
6. **Confirm.** The next complete scan that no longer finds it marks the fix as done, in the remediation feed.

What Guardian does not do. It does not attack anything (the web scan only reads), it does not sign in to the applications, it does not judge a library safe because nobody has reported it yet (unknown flaws exist), and it never stores a secret: secrets found in a repository are recorded as rule, file and line, with the value hidden.

16 · Glossary and acronyms

Advisory	Official notice describing a vulnerability, affected and fixed versions, severity.
Attack vector	The way an attacker reaches the flaw: network, adjacent network, local, physical.
Build tool	A library used to build or test the application, never shipped to users.
CIA triad	Confidentiality, integrity, availability: the three things an attack can damage.
CISA	Cybersecurity and Infrastructure Security Agency (US). Keeps KEV, promotes VEX and SSVC.
CNA	CVE Numbering Authority: an organisation allowed to assign CVE numbers.
CVE	Common Vulnerabilities and Exposures: the global name of a vulnerability, CVE-YEAR-NUMBER.
CVSS	Common Vulnerability Scoring System: severity from 0 to 10, built from the vector.
CWE	Common Weakness Enumeration: the catalogue of types of mistakes.
Dependency (direct / transitive)	A library we chose / a library that came in through another one.
EPSS	Exploit Prediction Scoring System (FIRST): probability of exploitation in the next 30 days.
Exploit	The method or code that uses a vulnerability.
Configuration drift	Settings that slowly move away from the agreed safe baseline.
Penetration test	An authorised, real attack on an application to find what scanners miss.
Threat modelling	Thinking through, at design time, how each part could be abused.
False positive	A finding that is not really there; always recorded with its reason.
FIRST	Forum of Incident Response and Security Teams: maintains CVSS and EPSS.
GHSA	GitHub Security Advisory id.
Guardrail	A rule that forbids lowering a rating, whatever the other evidence says.
KEV	Known Exploited Vulnerabilities catalog (CISA): flaws already used in attacks.
NVD	National Vulnerability Database (NIST): CVEs with scores and weakness types.
OSV	Open Source Vulnerabilities: format and database that maps advisories to exact package versions.
OWASP	Open Worldwide Application Security Project: community that publishes the Top 10.
Reachability	Whether the vulnerable code can actually be reached in our application.
SBOM	Software Bill of Materials: the list of components and versions.
Severity / risk rating	How bad a flaw is in general / how urgent it is for us.
SSVC	Decision tree (CISA) giving an action: Track, Track*, Attend, Act.
TLS	Transport Layer Security: the encryption of https.
VEX	Vulnerability Exploitability eXchange: standard statement of affected / not affected, with justification.

Acronyms

ACL	Access Control List: the list of who may read, write or run a file or resource.
AV, AC, PR, UI	CVSS metrics: Attack Vector, Attack Complexity, Privileges Required, User Interaction (chapter 8).
BOD	Binding Operational Directive: an order from CISA to US federal agencies (BOD 22-01 created KEV).
CAA	Certification Authority Authorization: a DNS record listing who may issue certificates for a domain.
CERT	Computer Emergency Response Team.
CIA	Confidentiality, Integrity, Availability.
CIS	Center for Internet Security, which publishes the CIS Benchmarks: safe settings for systems.
CISA	Cybersecurity and Infrastructure Security Agency (US).
CNA	CVE Numbering Authority.
CORS	Cross-Origin Resource Sharing: which other sites may read a site's answers.
CPE	Common Platform Enumeration: standard names for products in NVD.
CSP	Content Security Policy: the header that lists which scripts may run.
CSPM	Cloud Security Posture Management: tools that check cloud settings.
CVE	Common Vulnerabilities and Exposures.
CVSS	Common Vulnerability Scoring System.
CWE	Common Weakness Enumeration.
DAST	Dynamic Application Security Testing: attacking the running application from outside.
DMARC	Domain-based Message Authentication, Reporting and Conformance: what to do with forged e-mail.
EASM	External Attack Surface Management: finding and checking everything reachable from the internet.
EPSS	Exploit Prediction Scoring System.
FIRST	Forum of Incident Response and Security Teams.
GHSA	GitHub Security Advisory.
HSTS	HTTP Strict Transport Security.
IAM	Identity and Access Management: accounts, roles and permissions.
KB	Knowledge Base article: the number of a Microsoft patch (e.g. KB5031354).
KEV	Known Exploited Vulnerabilities.
MFA	Multi-Factor Authentication: a second proof besides the password.
NIST	National Institute of Standards and Technology (US), which runs NVD.
NVD	National Vulnerability Database.
OSV	Open Source Vulnerabilities.
OWASP	Open Worldwide Application Security Project.

RCE	Remote Code Execution: an attacker runs their own code on the target.
RLS	Row-Level Security: database rules that decide which rows each user may see (used by Supabase).
SAST	Static Application Security Testing: reading the source code to find flaws (Guardian uses Semgrep).
SBOM	Software Bill of Materials.
SCA	Software Composition Analysis: listing the libraries an application uses and matching them to known vulnerabilities.
SPF	Sender Policy Framework: which servers may send e-mail for a domain.
SQL	Structured Query Language: the language used to talk to most databases.
SSRF	Server-Side Request Forgery: tricking a server into calling an address the attacker chooses, often an internal one.
SSVC	Stakeholder-Specific Vulnerability Categorization.
TLS	Transport Layer Security.
VEX	Vulnerability Exploitability eXchange.
XSS	Cross-Site Scripting: the attacker's script runs in another user's browser through the site.

17 · References

Every fact in this guide comes from these pages. The last column says how it was checked: **read in the original** means the text of the official page was compared with this guide; **checked through several sources** means the official page could not be read automatically (cisa.gov refuses automated readers, cve.org needs a browser) and the fact was confirmed by more than one independent source quoting it; **used by Guardian** means Guardian reads it at every scan.

#	Source	How it was checked
1	CVE Program · overview https://www.cve.org/About/Overview	checked through several sources
2	CVE Program · terminology: CVE Record, Reserved, Published, Rejected https://cve.mitre.org/blog/December112020_CVE_Program_Terminology_Updated-CVE_Record_Top-Level_Root_and_More.html	checked through several sources
3	NIST NVD · the CVE process https://nvd.nist.gov/general/cve-process	checked through several sources
4	FIRST · CVSS v3.1 specification https://www.first.org/cvss/v3.1/specification-document	read in the original, 5 Nov 2026
5	FIRST · CVSS v4.0 specification https://www.first.org/cvss/v4.0/specification-document	read in the original, 5 Nov 2026
6	FIRST · EPSS https://www.first.org/epss/	read in the original, 5 Nov 2026
7	FIRST · EPSS API https://api.first.org/data/v1/epss	used by Guardian at every scan
8	CISA · Known Exploited Vulnerabilities catalog https://www.cisa.gov/known-exploited-vulnerabilities-catalog	checked through several sources
9	CISA · Reducing the significant risk of known exploited vulnerabilities (KEV criteria) https://www.cisa.gov/known-exploited-vulnerabilities	checked through several sources
10	CISA · KEV criteria and process update https://www.cisa.gov/news-events/alerts/2022/06/07/cisa-provides-criteria-and-process-updates-kev-catalog	checked through several sources
11	CISA · Binding Operational Directive 22-01 https://www.cisa.gov/binding-operational-directive-22-01	checked through several sources
12	CISA · KEV data feed (JSON) https://www.cisa.gov/sites/default/files/feeds/known_exploited_vulnerabilities.json	used by Guardian at every scan
13	CISA · Minimum requirements for VEX https://www.cisa.gov/sites/default/files/2023-04/minimum-requirements-for-vex-508c.pdf	checked through several sources
14	CISA · VEX status justifications https://www.cisa.gov/sites/default/files/2023-01/VEX_Status_Justification_Jun22.pdf	checked through several sources
15	CISA · SSVC https://www.cisa.gov/ssvc	checked through several sources
16	MITRE · About CWE https://cwe.mitre.org/about/index.html	read in the original, 5 Nov 2026

17	MITRE · CWE-79 cross-site scripting https://cwe.mitre.org/data/definitions/79.html	checked through several sources
18	OWASP · Top 10:2025 https://owasp.org/Top10/2025/	read in the original, 5 Nov 2026
19	GitHub · About the GitHub Advisory Database https://docs.github.com/en/code-security/security-advisories/working-with-global-security-advisories-from-the-github-advisory-database/about-the-github-advisory-database	read in the original, 5 Nov 2026
20	GitHub · Advisory Database https://github.com/advisories	read in the original, 5 Nov 2026
21	OSV.dev https://osv.dev/	checked through several sources
22	OpenSSF · OSV schema https://ossf.github.io/osv-schema/	read in the original, 5 Nov 2026
23	OSV.dev · data quality https://google.github.io/osv.dev/data_quality.html	checked through several sources
24	MDN · HTTP security headers https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers	checked through several sources
25	Mozilla · Server side TLS guidelines https://wiki.mozilla.org/Security/Server_Side_TLS	checked through several sources
26	testssl.sh https://github.com/testssl/testssl.sh	checked through several sources
27	RFC 9116 · security.txt https://www.rfc-editor.org/rfc/rfc9116	checked through several sources
28	RFC 7208 · SPF; RFC 7489 · DMARC; RFC 8659 · CAA https://www.rfc-editor.org/rfc/rfc7489	checked through several sources