Software Marketplaces for Extensible Web Apps

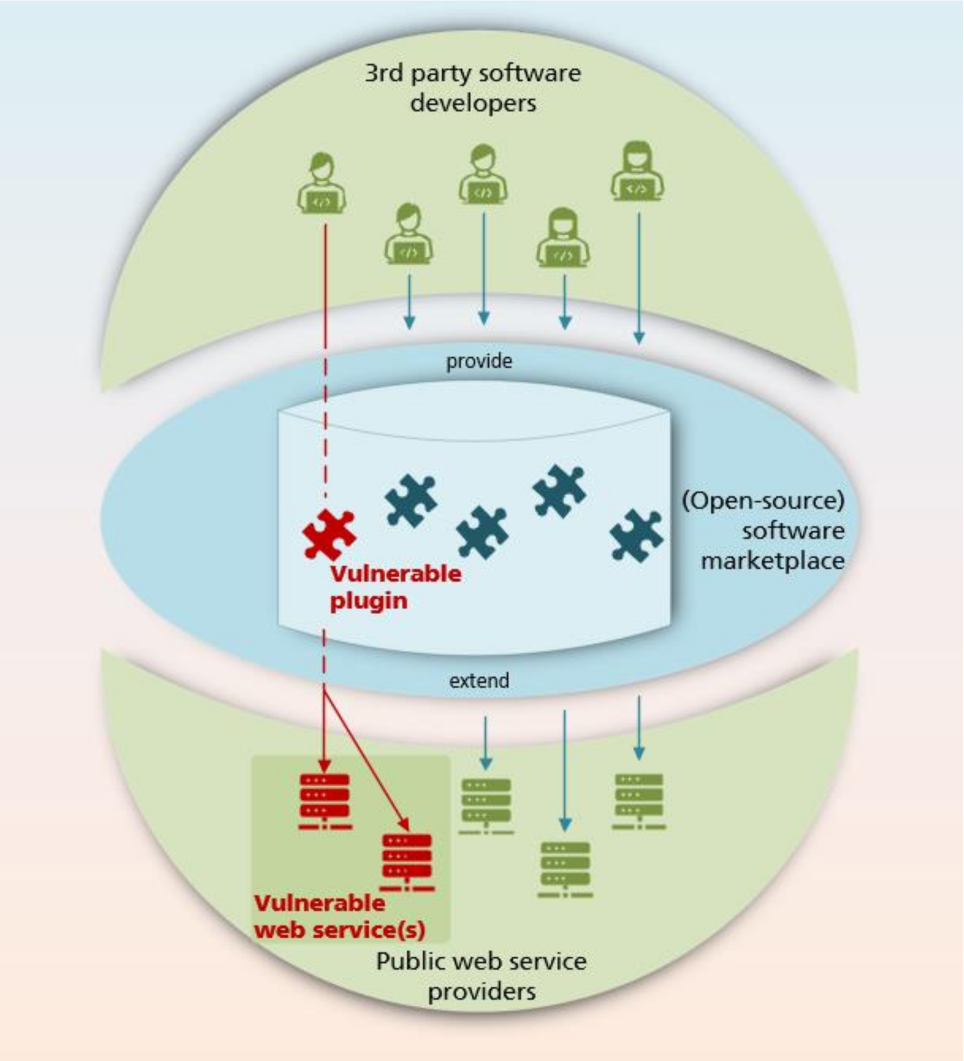
Curse and Blessing for Security Research

Software marketplaces can pose a high security risk:

- (Almost) anyone can contribute…
- … with code that is actually executed
- Establishing quality gates is difficult and costly
- Vulnerabilities potentially affect a large number of users

Software marketplaces also give opportunities for research:

- Often share the same technology platform (libraries, frameworks) which facilitates pattern-based searching
- May provide large data sets which again can be used as evaluation baseline: **Vulnerability history**


3rd party software developers
provide
(Open-source) software marketplace
Vulnerable plugin
extend
Vulnerable web service(s)
Public web service providers

Software Marketplaces for Extensible Web Apps

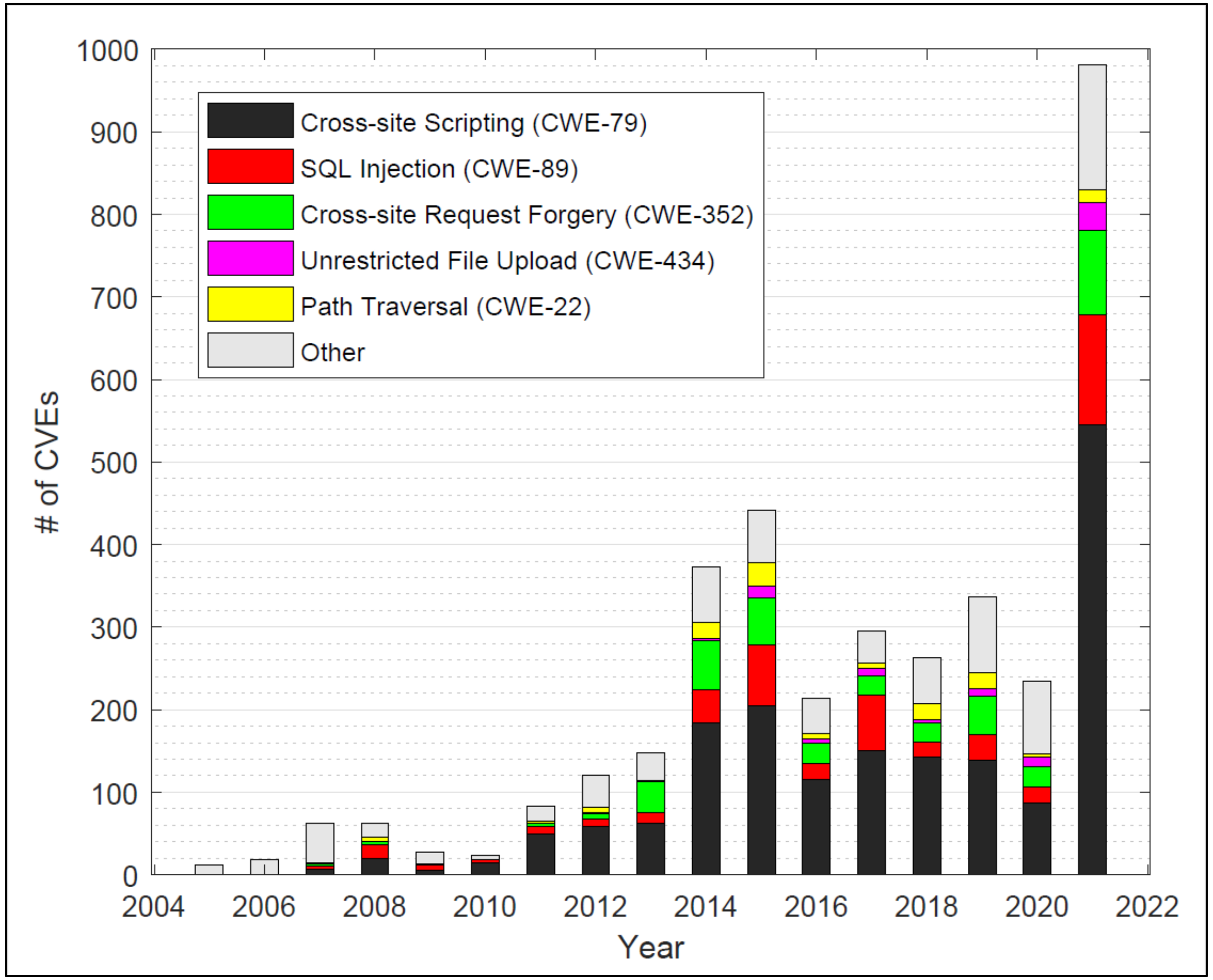
Vulnerability histories: What can we learn from the past?

In case of WordPress we found that…

- … 98.97% of all WordPress vulnerabilities in 2021 are caused by 3rd party plugins from the plugin store
- … more than 84.6% of all vulnerabilities are related to just five types of improper input validation vulnerabilities:
  - Cross-site scripting
  - SQL injection
  - Cross-site request forgery
  - Unrestricted file upload
  - Path traversal
- … the overall disclosed vulnerabilities reached a peak of 971 CVE entries in 2021

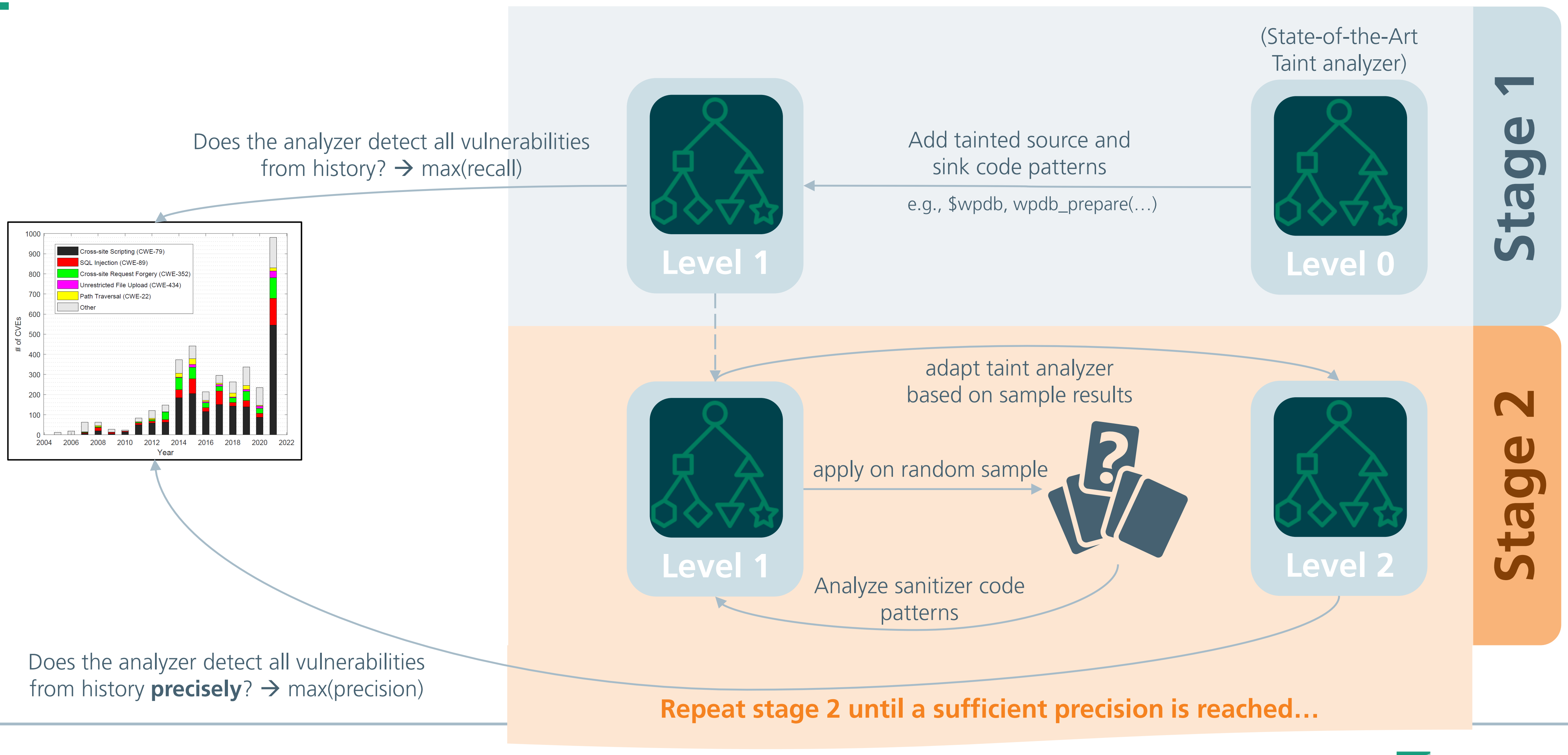
→ **If a »perfect« code analysis tool could detect all user input vulnerabilities (= 84.6% of 2021 WordPress vulns!) from a given <u>history</u>, would it detect <u>new</u> vulnerabilities as well?**

→ **How does tweaking precision and recall in the history affect the performance in the wild?**



Idea: Systematic Specialization of Taint Analyzers

»Training« the Taint Analyzer with the Vulnerability History (of a Software Marketplace)



(State-of-the-Art Taint analyzer)

Add tainted source and sink code patterns

e.g., $wpdb, wpdb_prepare(…)

Does the analyzer detect all vulnerabilities from history? → max(recall)

adapt taint analyzer based on sample results

apply on random sample

Analyze sanitizer code patterns

Does the analyzer detect all vulnerabilities from history **precisely**? → max(precision)

Repeat stage 2 until a sufficient precision is reached…