

### What Is This Bar Graph About?

The graph compares how well three tools—Oyente, Slither, and Solhint—perform when checking 100 smart contracts for security issues. Each bar shows how good or bad the tool is in a specific area.

#### Metrics (What Each Category Means)

##### 1. Accuracy (%)

> How often the tool gives the correct result.

Slither (90%): Almost always gives the right answer.

Oyente (70%): Sometimes right, sometimes misses things.

Solhint (65%): More focused on code style than deep security issues.

##### 2. Speed (Contracts per Minute)

> How many smart contracts it can scan quickly.

Slither (85): Very fast—good for large projects.

Solhint (75): Also fast, because it checks for simpler things.

Oyente (35): Slower because it uses heavy analysis techniques.

##### 3. Vulnerability Detection

> How many different types of bugs or problems the tool can find.

Slither (95): Excellent—it catches most known vulnerabilities.

Oyente (68): Catches several common bugs but not all.

Solhint (60): Focused on best practices, not deep bugs.

#### 4. False Positive Rate (%)

> When the tool says “there is a problem” but there actually isn’t.

Lower numbers are better here.

Slither (10%): Rarely gives wrong warnings.

Oyente (30%): Gives some incorrect alerts.

Solhint (40%): Might report many small issues that aren’t real bugs.

#### 5. Ease of Use

> How easy it is for developers to install and use the tool.

Solhint (85): Very user-friendly, especially for beginners.

Slither (75): Slightly technical but well-documented.

Oyente (60): Harder to set up and use for new developers.

### Performance Metrics & Tool Comparison

Metric	Oyente	Slither	Solhint	Best Tool in Metric
Accuracy (%)	70%	90%	65%	✅ Slither
Speed (contracts/min)	35	85	75	✅ Slither
Vulnerability Detection	68%	95%	60%	✅ Slither
False Positive Rate (%)	30%	10%	40%	✅ Slither (lower is better)
Ease of Use	60	75	85	✅ Solhint

---

### Vulnerabilities Not Detected by Tools

Here's a breakdown of commonly **missed vulnerabilities** by each tool based on public documentation and performance evaluations:

---

#### Oyente – Missed or Poorly Detected Vulnerabilities

- **Re-entrancy:** Detects simple cases, but fails in complex inter-contract calls or proxy patterns.
- **Unchecked Low-Level Calls:** Often missed when `call.value()` is used without `.transfer()` or `.send()`.
- **Integer Overflows/Underflows:** Doesn't fully support detection in newer Solidity versions (post-0.8.x).
- **Timestamp Dependence:** Limited detection or inaccurate flagging.

- **No support for recent Solidity constructs** (e.g., custom errors, modifiers with return values).
- 

#### **Slither – Missed or Limited Detection**

- **Complex Reentrancy**: Can miss deeply nested or delegate-based reentrancy attacks.
- **Gas Limit & Loops**: Doesn't analyze gas consumption patterns for DoS via block gas limit.
- **Front-Running (Tx Order Dependence)**: Does not provide complete support.
- **Oracle Manipulation**: No analysis for dependency on off-chain data feeds.

Still, **Slither is the most comprehensive among the three** and regularly updated with new detectors.

---


































#### **Solhint – Missed or Not Designed to Detect**

- **Solhint is primarily a linter, not a static analyzer.**
  - It does **not detect vulnerabilities** like:
    - Reentrancy
    - Integer Overflow/Underflow
    - Denial of Service
    - Timestamp manipulation
  - Focus is more on **style issues, best practices, and code formatting.**
- 

#### **Summary & Recommendation**

- **Best Overall Tool: Slither**, due to high accuracy, speed, low false positives, and robust detection.
- **Oyente**: Older tool, less accurate, higher false positives, and slower.
- **Solhint**: Best for coding standards and ease of use but **not for security analysis.**

## Vulnerability Detection Comparison Table

Vulnerability Type	Oyente	Slither	Solhint	Remarks
Reentrancy	 Partial	 Good	 No	Oyente misses complex patterns; Slither detects many but not all cases.
Integer Overflow/Underflow	 Partial	 Yes	 No	Oyente struggles with Solidity $\geq 0.8.0$ ; Slither uses symbolic analysis.
Timestamp Dependence	 Limited	 Yes	 No	Slither flags it clearly. Oyente misses edge cases.
DoS with Block Gas Limit	 No	 Limited	 No	Slither only partially warns; others don't detect it.
Unchecked Low-Level Calls	 Some	 Yes	 No	Slither detects call, delegatecall misuse; Oyente not reliable.
Front-running (Tx Order)	 No	 Limited	 No	Only some patterns flagged by Slither.
Access Control Flaws	 No	 Yes	 No	Only Slither checks onlyOwner or similar.
Uninitialized Storage Pointers	 No	 Yes	 No	Slither flags this clearly.
Shadowing/Inconsistent Naming	 No	 Yes	 Partial	Solhint handles naming conventions only; Slither detects actual issues.
Style Issues / Best Practices	 No	 Limited	 Yes	Solhint is best suited for this.
Gas Optimization Issues	 No	 Yes	 No	Only Slither has detectors for gas cost inefficiencies.

---

## Report Summary: Tool-Wise

### ◆ 1. Oyente

- **Strengths:**
  - Detects basic reentrancy and overflow/underflow.
- **Weaknesses:**
  - Poor speed (35 contracts/min).
  - High false positive rate (30%).
  - Misses DoS, timestamp, access control vulnerabilities.

- Not updated for newer Solidity versions.

## ◆ 2. Slither

- **Strengths:**
  - High detection rate across most vulnerability types.
  - Fast (85 contracts/min), highly accurate (90%), and low false positives (10%).
  - Regularly updated with new vulnerability detectors.
- **Limitations:**
  - Partial coverage of DoS via gas limits and front-running scenarios.
  - May miss extremely complex nested or proxy-based reentrancy.

## ◆ 3. Solhint

- **Strengths:**
  - Excellent for code **style checking**, naming conventions, and formatting.
  - Most user-friendly (Ease of Use score: 85).
- **Limitations:**
  - **Not meant for security analysis.**
  - Fails to detect critical vulnerabilities like reentrancy, overflows, DoS, etc.

---

### Recommendation

If your goal is **security analysis of smart contracts**, **Slither is the best choice** among the three. Use **Solhint** additionally for **style and readability**, and **avoid relying solely on Oyente** unless you're comparing legacy behavior.