Cyber Advisory: react2shell vulnerability

1. What is react2shell?

The **react2shell vulnerability** refers to a critical remote code execution (RCE) vulnerability in modern JavaScript web frameworks (especially **React** and **Next.js**). Attackers can **execute malicious code on the server** that provides the web application through manipulated inputs or insecure build configurations. In short: Seemingly harmless user data becomes executable code – a **complete system takeover** is possible.

Related CVEs:

CVE	Criticality	Affected Systems
CVE-2025-55182 (React)	Critical (CVSS **10.0**)	React 19.x (Server Components); affects numerous frameworks (e.g. Next.js)
CVE-2025-66478 (Next.js)	Critical (CVSS **10.0**)	Next.js 15.x/16.x (with React Server Components / "App Router")

2. Why is react2shell relevant?

React2shell poses a significant cyber risk.

Three factors in particular are decisive:

In practice, react2shell has already been **actively exploited worldwide**. Several organizations have become victims within days of becoming known. The US agency CISA immediately listed the vulnerability as a "Known Exploited Vulnerability" and obliged federal authorities to install appropriate updates by **December 26, 2025** – a clear warning signal regarding the urgency and seriousness of the situation.

3. Typical attack scenarios

React2shell can manifest itself in different vulnerability scenarios. Typical attack routes are, for example:

1. **Infected user input:** For example, specially prepared JSON data or query parameters that are processed unchecked by the build or server process of the React app. This will execute **unwanted code**.
2. **Manipulated third-party components:** Attackers inject malicious code into dependent libraries or npm packages. If these are used in the build pipeline, **the build server is compromised** (supply chain attack).
3. **Malicious file upload:** A file (such as a seemingly legitimate configuration file) is uploaded to the web application and analyzed by the system. A gap in the processing process opens a **shell in the background** and executes code.

Result: In all cases, there is often a risk **of complete takeover** of the affected server, often followed by further intrusion into the network (*lateral movement*).

4. Which companies are particularly at risk?

Basically, every company with modern web applications is vulnerable. However, companies that use **React-based systems** and **do not have a mature security and patch management** system are particularly at risk. Some examples:

Risk	Example companies	Risk
React-based customer portals	Banks, Commerce	**High**
CI/CD automation without separation (build vs. prod)	Medium-sized companies, startups	**High**
Use of numerous npm packages	E-commerce companies, platforms	**High**
Low DevSecOps maturity	Small businesses (SMEs)	**Very high**

(Explanation: **High** = increased risk of attack; **Very high** = acute risk of attack)SMEs in particular often underestimate the danger, as they use modern frameworks but do not have appropriate security processes.

5. Impact

If the react2shell vulnerability is successfully exploited, the following non-conclusive damage can occur:

- **Forensics & Recovery Costs:** Investigating compromised servers, cleaning up systems, restoring from backups, and rebuilding infrastructure.
- **Data breaches (GDPR fines):** If customer data or confidential information has been leaked, there is a risk of fines from the supervisory authorities.
- **Business interruption:** If the build pipeline or web service is crippled (e.g., by ransomware or mitigation shutdown), critical business processes can be down for several days. This results in loss of income and additional costs.
- **Extortion / ransom demands:** In the event of a takeover, attackers can often threaten to publish stolen data or encrypt the company.

6. Risk assessment: Important questions for companies

In order to be able to assess the risk, you should ask yourself specific questions about the react2shell problem:

- Do you use **React**, **Node.js** or comparable web frameworks in your applications?
- Is your **CI/CD pipeline** segmented – i.e. build and production systems strictly separated?
- How do you handle library updates? Are **npm dependencies** regularly updated and checked for vulnerabilities via *dependency scanning* (e.g. Snyk, OWASP dependency check)?
- Are you creating a **software bill of materials (SBOM)** and monitoring your software supply chain for security vulnerabilities?
- Do you perform regular **static code analysis** and **penetration testing** of your web applications?
- Do you have an **incident response plan** specifically for web application security incidents (e.g., emergency response plan in the event of an RCE attack)?

7. Recommended protective measures

1. **Immediate patch management:** Immediately update all affected systems (React/Next.js applications) to the latest security updates.

   Information on patching HERE:
   https://react.dev/blog/2025/12/03/critical-security-vulnerability-in-react-server-components

2. **Next-Gen Firewall (NGFW) / Web Application Firewall (WAF):** A firewall can block known attack calls and should be enabled for compromised web services (with special rules for React2shell patterns if necessary).

   First IP-based IOCs and BSI assessment HERE:
   - https://github.com/DataDog/indicators-of-compromise/tree/main/react-CVE-2025-55182
   - https://aws.amazon.com/de/blogs/security/china-nexus-cyber-threat-groups-rapidly-exploit-react2shell-vulnerability-cve-2025-55182/
   - https://www.cisa.gov/news-events/alerts/2025/12/05/cisa-adds-one-known-exploited-vulnerability-catalog

3. **Hardened CI/CD pipeline:** Strict privileging, isolation of build environments, and careful secrets management so that a compromised build server doesn't compromise the entire network.

4. **Dependency Scanning & SBOM:** Monitor the use of external libraries. Use tools such as Snyk or GitHub Dependabot and maintain a software bill of materials to quickly identify vulnerable components.

5. **Contingency plan:** Check your incident response plans for timeliness and usability.

8. Conclusion

React2shell is a **critical vulnerability in modern web technologies** that illustrates how vulnerable broad parts of the IT landscape can be. Many companies – especially smaller ones – are affected and underestimate the risk. Insurers are already recording increasing claims due to such attacks on popular web stacks.

You should proactively address the react2shell issue. A structured risk assessment helps to realistically assess exposure. At the same time, you should take advantage of this vulnerability and consider **additional security measures**. In this way, you also help to prevent or limit other risks and potential damage.

React2shell is therefore not only an alarm signal for developers and IT departments, but also a wake-up call for the insurance industry to continuously monitor cyber risks vigilantly and actively counteract them.