ON2IT AND AMS-IX PRESENT: THREAT TALKS

Threat Talks
Web Application Threats

threat-talks.com

When business logic becomes an attack surface

Web applications were built to move fast. To connect users, data, and services at scale. They sit at the center of modern business, handling authentication, payments, customer data, and core workflows. In many organizations, they are the business.

That central role comes with risk. Web applications are publicly reachable, constantly changing, and assembled from layers of frameworks, APIs, and third-party components. Each layer adds complexity. Each dependency adds trust. Attackers don't need to breach the network when the application itself is exposed by design.

As web apps grow more dynamic, so do the threats against them. Flaws in input handling, authentication, or dependency management can turn a single request into data exposure, account takeover, or remote code execution. The challenge is no longer just building applications that work; it's ensuring the code running at the heart of your business can't be quietly turned against you.

In this Threat Talks infographic we will discuss the following threats:

- MongoBleed
- React2Shell

12% Basic web application attacks are involved in roughly 12% of breaches.

Source: Verizon

56% of organizations have experienced a web application breach in the past 12 months.

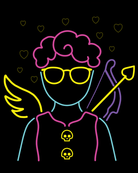
Source: Cybersecurity Insiders

60% Blind spots in application visibility leave up to 60% of organizations struggling to detect threats before they escalate.

Source: Fortinet

Last year, **311 billion** web application and API attacks occurred.

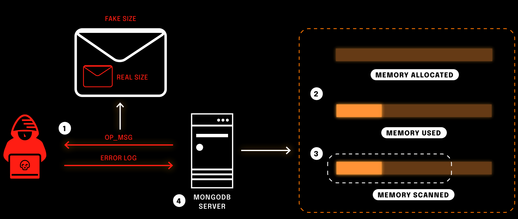
Source: Akamai



MongoBleed

CVE-2025-14847

On December 19, 2025, a critical vulnerability in MongoDB Server was publicly disclosed and assigned CVE-2025-14847, nicknamed MongoBleed. The flaw exists in the database's zlib compression handling code and lets an unauthenticated attacker remotely trigger the server to return uninitialized heap memory, potentially exposing sensitive data such as plaintext credentials, API keys, session tokens, and other in-memory artifacts..



1. Malicious compressed message

The attacker sends a malicious compressed MongoDB message that claims a much larger uncompressed size than the data actually contains, and includes a BSON string without a null terminator.

2. Buffer uninitialized

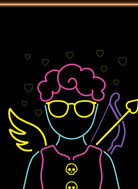
MongoDB allocates a memory buffer based on the claimed size, but after decompression only the real, smaller message is written, leaving the rest of the buffer uninitialized.

3. Null terminator

While parsing the BSON, MongoDB keeps reading memory past the end of the real message until it encounters a null terminator, unintentionally including adjacent heap memory.

4. Leaked memory

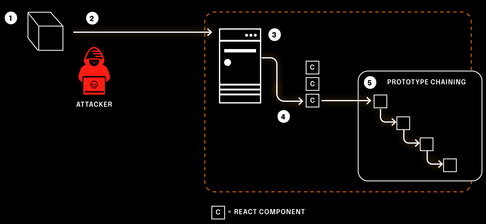
The malformed BSON triggers an error response, and MongoDB returns a verbose error message that includes the leaked memory contents.



React2Shell

Critical Unauthenticated RCE vulnerability in React.js and Next.js

React2Shell, tracked as CVE-2025-55182, is an unauthenticated remote code execution vulnerability in the React Server Components which is a commonly used function of the React.js framework. The trick with the vulnerability is that your project does not even need to use any React Server Function endpoints for it to be vulnerable. This is one of the causes for the 10.0 CVSS score as it requires no specific functionality. In combination with the widespread usage of the React.js framework (and other frameworks that use React, like Next.js) made this a very critical vulnerability.



1. Craft payload

1. The attacker crafts a simple HTTP POST Request with a multipart/form-data body consisting of serialized data for the react framework.

M Keep frameworks and dependencies up to date, disable unsafe deserialization patterns, validate and sanitize all user input, and perform secure code reviews to eliminate dangerous constructor or dynamic execution patterns.

2. Send payload to webserver (POST request)

2. The attacker delivers the malicious payload to the target webserver via a normal-looking HTTP POST request, attempting to blend in with legitimate traffic.

M Use a Web Application Firewall (WAF) (or other layer 7 compatible firewall) to inspect and block suspicious payloads, enforce strict content-type validation, apply rate limiting, and monitor logs for anomalous request patterns.

3. Webserver forwards payload to vulnerable React components

3. The attacker relies on the backend application logic to forward the crafted payload to a vulnerable React component that processes serialized data without sufficient validation.

M Apply strict server-side validation before passing data to components, implement schema validation (e.g., JSON schema), enforce least-privilege design between services, and use runtime application self-protection (RASP) to detect unsafe object manipulation.

4. React component deserializes payload (Prototype Chaining)

4. During deserialization, the attacker abuses prototype chaining to manipulate object inheritance, injecting properties that lead toward execution of arbitrary code paths.

M Prevent prototype pollution by freezing or sealing objects where possible, avoid merging untrusted objects into application state, use safe deserialization libraries, and enable dependency scanning tools to detect known vulnerable packages.

5. Constructor or Function execution (Code/Payload execution)

5. The manipulated object chain results in invocation of a constructor or dynamic function call, allowing execution of the attacker-controlled JavaScript payload and potentially leading to shell access.

M Disable use of dynamic code execution (e.g., eval, new Function), enforce Content Security Policy (CSP), deploy endpoint detection and response (EDR) solutions, run applications with least privilege, and continuously monitor for abnormal process or command execution behavior.

threat-talks.com

ON2IT ZERO TRUST INNOVATORS

amsix

CONTACT US