Technical Report

Inspektor Gadget

Security Assessment

Prepared for:
OSTIF

SHIELDER
WEB SECURITY

1. Document Details

Classification	**Public** - CC BY-SA 4.0
Last review	April 22, 2026
Author	Pietro Tirenna, Nicolò Daprelà

1.1. Version

Identifier	Date	Author	Note
v1.0	February 11, 2026	Pietro Tirenna, Nicolò Daprelà	First version
v1.1	February 26, 2026	Davide Silvetti	Peer review
v1.2	April 22, 2026	Pietro Tirenna	Public release

1.2. Contacts Information

Company	Name	Position	Contact
Shielder	Abdel Adim `Smaury` Oisfi	CEO	abdeladim.oisfi@shielder.it
Shielder	Pietro Tirenna	Consultant	pietro.tirenna@shielder.com
Shielder	Nicolò Daprelà	Consultant	nicolo.daprela@shielder.com
Shielder	Davide Silvetti	Consultant	davide.silvetti@shielder.com
OSTIF	Derek Zimmer	Executive Director	derek@ostif.org
OSTIF	Amir Montazery	Managing Director	amir@ostif.org
OSTIF	Helen Woeste	Communications and Community Manager	helen@ostif.org
OSTIF	Tom Welter	Project Manager	tom@ostif.org
Inspektor Gadget	Francis Laniel	Core Developer	flaniel@microsoft.com
Inspektor Gadget	Mauricio Vasquez	Core Developer	mauriciov@microsoft.com

1.3. About OSTIF

The **Open Source Technology Improvement Fund (OSTIF)** is dedicated to resourcing and managing security engagements for open source software projects through partnerships with corporate, government, and non-profit donors. We bridge the gap between resources and security outcomes, while supporting and championing the open source community whose efforts underpin our digital landscape.

Over the past ten years, OSTIF has been responsible for the discovery of over 800 vulnerabilities, (121 of those being Critical/High), over 13,000 hours of security work, and millions of dollars raised for open source security. Maximizing output and security outcomes while minimizing labor and cost for projects and funders has resulted in partnerships with multi-billion dollar companies, top open source foundations, government organizations, and respected individuals in the space. Most importantly, we've helped over 120 projects and counting improve their security posture.

Our directive is to support and enrich the open source community through providing public-facing security audits, educational resources, meetups, tooling, and advice. OSTIF's experience positions us to be able to share knowledge of auditing with maintainers, developers, foundations, and the community to further secure our infrastructure in a sustainable manner.

We are a small team working out of Chicago, Illinois. Our website is ostif.org. You can follow us on social media to keep up to date on audits, conferences, meetups, and opportunities with OSTIF, or feel free to reach out directly at contactus@ostif.org or our Github.

Derek Zimmer, Executive Director
Amir Montazery, Managing Director
Helen Woeste, Communications and Community Manager
Tom Welter, Project Manager

2. Summary

3. Executive Summary

The document aims to highlight the findings identified during the “Security Assessment” against the "Inspektor Gadget" product described in section “3.2 Context and Scope”.

For each detected findings, the following information are provided:

- **Severity**: findings score ("3.7 Findings Severity Classification").
- **Affected resources**: vulnerable components.
- **Status**: remediation status ("3.8 Remediation Status Classification").
- **Description**: type and context of the detected finding.
- **Impact**: loss of confidentiality, data integrity and/or availability in case of a successful exploitation and conditions necessary for a successful attack.
- **Proof of Concept**: evidence and/or reproduction steps.
- **Suggested remediation**: configurations or actions needed to mitigate the finding.
- **References**: useful external resources.

3.1. Overview

From January 12, 2026 to February 6, 2026 **Shielder** was hired by the **Open Source Technology Improvement Fund (OSTIF)** to perform a *Security Audit* of **Inspektor Gadget**, a set of libraries and tools for data collection and inspection on Kubernetes clusters and Linux hosts.

Inspektor Gadget is both a framework and a toolkit to enhance observability on a Linux machine/Kubernetes node, using the eBPF technology. Inspektor Gadget manages the packaging, deployment and execution of "gadgets", which are essentially eBPF programs encapsulated in OCI images. Gadgets export events that are caught by the tool and that can be filtered, sorted, exported or enriched.

A team of 2 (two) Shielder researchers worked on this project for a total of 35 person-days of effort.

3.2. Context and Scope

Inspektor Gadget is built around the following core components:

- CLI tools, such as `ig`, `kubectl-gadget` and `gadgetctl` that are used to initiate/interact with gadgets.
- Operators, the modules of the execution pipeline. They are modular components used to implement the processing chain. They are responsible for specific tasks, such as loading eBPF programs, enriching events, sorting output, etc.
- Runtime: the system that actually runs the pipeline, by parsing parameters and orchestrating operators. In Kubernetes, it is constituted by a DaemonSet pod deployed on the node. On Linux hosts, it can either be executed locally (`ig run`) or deployed as a service (`ig daemon`).
- Gadgets: OCI images that package eBPF source code and metadata (or, optionally, WASM post-processing modules). While users can implement their own gadget, a list of official gadgets is provided by the project.
- Distribution/Security: the component that pulls OCI images from registries, verifies image signature and enforces restrictions on what gadgets can be executed.

For what concerns the attack surface in scope, with the support of Inspektor Gadget maintainers, the Shielder team have modeled the following main attack scenarios:

- An unauthorized or authorized attacker, able to contact a deployed Inspektor Gadget instance, might abuse its functionalities for privilege escalation purposes.
- A malicious or compromised container, running in a host where Inspektor Gadget is deployed, might abuse its functionalities for privilege escalation purposes.
- In a scenario where the observability granted by Inspektor Gadget is leveraged for security monitoring (e.g. tracing process executions on the host) a compromised process or container might find ways to bypass tracing.
- In Kubernetes deployments, permissions granted to Inspektor Gadget operators, or to the DaemonSet pod, would allow an attacker to escalate their privileges in the cluster.

It's important to notice that the security of the eBPF validator and the WASM runtime themselves is out-of-scope of the present security audit and outside of Inspektor Gadget threat model.

3.3. Methodology

The activity was carried out following the standard Shielder security audit methodology, which focuses on discovering:

- Vulnerabilities: findings that can be immediately exploited by an attacker to compromise the **CIA** (confidentiality, integrity, availability) properties of the system.
- Hardening recommendations: properties/patterns that, if implemented, would reduce the attack surface of the system.

Due to the typically constrained resources of Open Source projects, it is critical to focus on actionable and reasonable issues that, upon remediation, immediately increase the security posture of the project.

For this audit, the following techniques and approaches were employed:

- Collaborative threat modeling, with the goal of identifying the critical parts of the attack surface.
- Manual source code review, focused on the most exposed components, such as the exposed `ig daemon` deployment.
- Dynamic experiments on dedicated labs, with the goal of reproducing potential issues or fuzzing the system with edge cases.
- Static Analysis using dedicated SAST tools, such as semgrep and gosec.
- AI-assisted source code review, focused on reducing the time needed to understand the code, or quickly explore less-critical portions of the codebase to increase the audit coverage.

In order to dynamically experiment with the tool, three labs were setup by the researchers' team:

- Local deployment of Inspektor Gadget by means of `sudo ig run` on a Linux host.
- Remote deployment of Inspektor Gadget by means of `sudo ig daemon` on the host, and `gadgetctl` from another machine.
- Kubernetes deployment of Inspektor Gadget on a `minikube` cluster, gadget installed via `krew`.

3.4. Audit Summary

The overall security posture of **Inspektor Gadget** is adequately mature from both a secure coding and design point of view, even though there is still room for further refinement in some areas.

The Shielder team identified a total of three vulnerabilities, two of them having Medium severity, and one with Low severity. Additionally, the audit produced six hardening recommendations.

The main threats are caused by an insufficient segregation of the elevated `root` permissions that the tool has over the system. This makes it quite interesting for attackers looking for paths to escalate privileges on a partially compromised system.

Moreover, the Shielder team audited the official gadgets, and has discovered six bypasses that would allow a compromised container to evade the tracing. Therefore, for users aiming to use Inspektor Gadget for security monitoring of Kubernetes clusters and Linux hosts, there are some limitations that slightly decrease the reliability of the tool.

3.5. Recommendations

Implement gRPC Authorization Layer

When deployed as a service, Inspektor Gadget is implemented as a gRPC server that exposes APIs to list gadgets, fetch metadata, run new gadgets, pull events, and in general interact with the runtime.

While mTLS-based authentication can be enabled for the server, there is currently no support for authorization: once a client can authenticate to the gRPC server, every API becomes accessible. This practice violates the *Principle of Least Privilege* and, due to the security-critical APIs that Inspektor Gadget supports, increases the attack surface for privilege escalation on a node/cluster.

It is recommended to implement an authorization layer to properly segregate roles: some users might just have permissions to "attach" to existing gadgets and query events, while other users, more privileged, have permission to pull and run new gadgets.

Enrich Security Section in Documentation

Currently, the security documentation of Inspektor Gadget contains information to report vulnerabilities to the project.

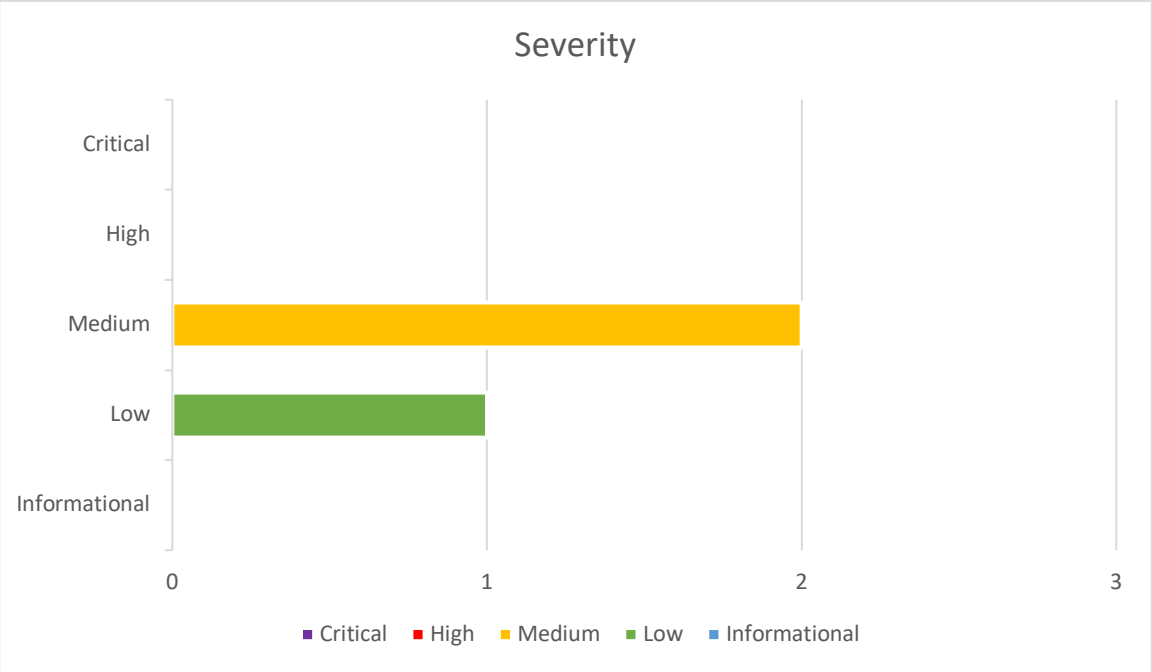
It is recommended to add more information about the security model of the project such as documenting how the system is secured by design, what residual risks remain, and how users should deploy safely.

The document should clearly highlight:

- How an attacker might abuse arbitrary eBPF to escalate their privileges on the host, and how even the official gadgets can be abused to leak sensitive data from the host.
- The controls that are available, such as image verification, the gadget allow-list, the pulling policy (it's worth mentioning that this information is already detailed throughout the documentation, so the security document should simply mention and reference the relevant docs).
- The known accepted risks, such as the extensive permissions granted to the RBAC role applied to the DaemonSet pod in Kubernetes.
- The known limitations of tracing, which would allow potential attackers to evade controls.

3.6. Results Summary

The following chart shows the number of findings found per severity:



ID	Finding	Severity	Status
1	Command Injection in ig image build	MEDIUM	Closed
2	Unsanitized ANSI Escape Sequences in Columns Output Mode	LOW	Closed
3	Denial of Service via Event Flooding	MEDIUM	Closed

3.7. Findings Severity Classification

Severity	Description
CRITICAL	Vulnerability that allows to compromise the whole application, host and/or infrastructure. In some cases, it allows access, in read and/or write, to highly sensitive data, totally impacting the resources in terms of confidentiality, integrity and availability. Usually, such vulnerabilities can be exploited without the need of valid credentials, without considerable difficulty and with the possibility of automated, highly reliable, and remotely triggerable attacks. Vulnerabilities marked with this severity must be resolved quickly, especially in production environment.
HIGH	Vulnerability that significantly affects the confidentiality, integrity, and availability of confidential and sensitive data. However, the prerequisites for the attack affect its likelihood of success, such as the presence of controls or mitigations and the need of a certain set of privileges.
MEDIUM	Vulnerability that allows to obtain only a limited or less sensitive set of data, partially compromising confidentiality. In some cases, it may affect the integrity and availability of the information, but with a lower level of severity. In addition, the chances of success of such vulnerability may depend on external factors and/or conditions outside the attacker's control.
LOW	Vulnerability resulting in a limited loss of confidentiality, integrity, and availability of data. In some cases, it depends on conditions not aligned to a real scenario or requires that the attacker has access to credentials with a high level of privileges. In addition, a low severity vulnerability may provide useful information to successfully exploit a higher impact vulnerability.
INFORMATIONAL	Problems that do not directly impact confidentiality, integrity, and availability. Usually, these problems indicate the absence of security mechanisms or the improper configuration of them. Mitigation of this type of problem increases the general level of security of the system and allows in some cases to prevent potential new vulnerabilities and/or limit the impact of existing ones.

3.8. Remediation Status Classification

Status	Description
Open	Vulnerability not mitigated or insufficient mitigation.
Not reproducible	Vulnerability not reproducible due to environment changes or to mitigation of other vulnerabilities required in the reproduction steps.
Closed	Vulnerability mitigated. The security patch applied is reasonably robust.

4. Findings Details

Analysis results are discussed in this section.

4.1. Command Injection in ig image build

Severity	MEDIUM
Affected Resources	cmd/common/image/build.go cmd/common/image/helpers/Makefile.build cmd/common/image/helpers/Makefile.build.btfgen
Status	Closed

Update

The vulnerability was reported through the GitHub Security section of the repository, in the linked advisory. It was fixed on release v0.51.1 and was assigned CVE-2026-24905.

Description

The `ig` binary provides a subcommand for image building, used to generate custom gadget OCI images. The subcommand accepts custom build options defined inside a YAML gadget manifest file or via command line options.

The functionality that handles build options is implemented in the `cmd/common/image/build.go`.

The following is the code responsible to construct the build command:

```
func buildCmd(options buildOptions) []string {
    cmd := []string{
        "make", "-f", filepath.Join(options.outputDir, "Makefile.build"),
        "-j", fmt.Sprintf("%d", runtime.NumCPU()),
        "OUTPUTDIR=" + options.outputDir,
        "CFLAGS=" + options.cFlags,
        "FORCE_COLORS=" + options.forceColorsFlag,
    }

    if options.ebpfSourcePath != "" {
        cmd = append(cmd, "EBPFSOURCE="+options.ebpfSourcePath, "ebpf")
    }
    if options.wasmSourcePath != "" {
        cmd = append(cmd, "WASM="+options.wasmSourcePath, "wasm")
    }
    if options.btfgen {
        cmd = append(cmd, "BTFHUB_ARCHIVE="+options.btfHubArchivePath,
"btfgen")
    }

    return cmd
}
```

The `Makefile.build` file is a Makefile template employed during the building process.

This file includes user-controlled data in an unsafe fashion, specifically some parameters are embedded without an adequate escaping in the commands inside the Makefile.

This implementation is vulnerable to command injection: an attacker able to control values in the `buildOptions` structure would be able to execute arbitrary commands during the building process.

Impact

An attacker able to exploit this vulnerability would be able to execute arbitrary commands:

- On the Linux host where the `ig` command is launched, if images are built with the `--local` flag.
- On the build container invoked by `ig`, if the `--local` flag is not provided.

Attack Complexity

The attacker would need a way to control either the full `build.yml` file passed to the `ig image build` command, or one of its options, or one of the command line flags which end up in the `buildOptions` structure.

Typically, this could happen in a CI/CD scenario that builds untrusted gadgets to verify correctness.

Related Issues

N/A

Proof of Concept

Vector: cflags

1. Create the file `build.yaml` with the following content:

```
ebpfsource: "program.bpf.c"
metadata: "gadget.yaml"
cflags: " ; touch poc.txt ; "
```

2. Create the file `gadget.yaml` with the following content:

```
name: test
description: test gadget
```

3. Create the file `program.bpf.c` with the following content:

```
#include <gadget/gadget.h>
char LICENSE[] SEC("license") = "GPL";
```

4. In the same directory where the files are run the command:

```
ig image build . -t test:latest
```

5. Notice that the file `poc.txt` gets created inside the directory.

Vector: ebpfsource, wasm

1. Create the file `build.yaml` with the following content:

```
ebpfsource: "$(shell touch poc1.txt)"
wasm: "$(shell touch poc2.txt)"
```

2. Create a file named `$(shell touch poc1.txt)`:

```
touch '$(shell touch poc1.txt)'
```

3. In the same directory where the files are run the command:

```
ig image build .
```

4. Notice that the files `poc1.txt` and `poc2.txt` get created inside the directory.

Vector: -o, --output

1. Create the file `build.yaml` with the following content:

```
wasm: dummy.go
```

2. Create the file `gadget.yaml` with the following content:

```
name: test
```

3. Create a directory named `$(shell touch poc3.txt)`:

```
touch '$(shell touch poc3.txt)'
```

4. Retrieve the full path of the created directory:

```
readlink -f '$(shell touch poc3.txt)'
```

5. In the same directory where the files are run the command replacing the `<PATH>` placeholder with the value retrieved at step 4:

```
ig image build . --local -o '<PATH>'
```

6. Notice that the file `poc3.txt` gets created inside the directory.

Vector: --btfhub-archive

1. Create the file `build.yaml` with the following content:

```
ebpfsource: test.c
```

2. Create the file `test.c`:

```
touch test.c
```

3. In the same directory where the files are run the command

```
sudo ig image build . --local --btfgen --btfhub-archive $(pwd)/'$(shell
touch poc4.txt)'
```

4. Notice that the file `poc4.txt` gets created inside the directory.

Suggested Remediations

Sanitize build options by providing a robust whitelist to filter on.

Alternatively, revisit the design of image building to prevent shell substitution inside of Makefiles.

References

- https://cwe.mitre.org/data/definitions/77.html
- https://cwe.mitre.org/data/definitions/78.html

4.2. Unsanitized ANSI Escape Sequences in Columns Output Mode

Severity	LOW
Affected Resources	pkg/columns/formatter/textcolumns
Status	Closed

Update

The vulnerability was reported through the GitHub Security section of the repository, in the linked advisory. It was fixed on release v0.49.1 and was assigned CVE-2026-25996.

Description

Inspektor Gadget can display events in the terminal by using custom output formatter, for example `columns`.

When string fields from eBPF events are rendered to the terminal in `columns` output mode `ig` does not perform any sanitization of control characters or ANSI escape sequences. This allows events to inject escape sequences that could be interpreted by the terminal emulator used to display the output.

It's important to notice that the `columns` output mode is the default when running `ig run` interactively.

Impact

The impact depends on the injection point, length limitations, and on the terminal used by the operator when running displaying `columns` output.

At the very least, the injection can be used for Log Injection, by inserting new lines or deleting existing ones.

However, by leveraging Operating System Command (OSC) ANSI escape sequences, the impact on modern terminal can vary, possibly allowing an attacker to:

- Write to the system clipboard.
- Create hyperlinks to attacker-controlled servers.
- Change window title.
- Potentially execute code (see referenced resources).

Attack Complexity

In order to inject ANSI codes, an attacker would need a way to control text that is going to be displayed in the `ig` terminal.

The most plausible scenario is that of an attacker that controls the code executed in a container observed by `ig`. At that point, it would be sufficient to arbitrarily change the value

of the `comm` attribute to inject in any case, independently of the gadget used by the `ig` operator.

Depending on the gadget, other injection points can be found. For instance, filenames can be used to inject into the `trace_open` gadget output.

Related Issues

N/A

Proof of Concept

1. Setup a Linux host and build/install `ig` version `0.48.0`
2. Run the attached `run_poc2.sh` on a terminal
3. Run `sudo ig run trace_open -c poc-escape-inject` on another terminal
4. Press "Enter" on the terminal attached to `run.sh`
5. Observe the events traced by `ig`
6. Notice that, at some point, the line where `/etc/shadow` is logged is overwritten `/etc/bashrc`, demonstrating the log injection

Suggested Remediations

Output meant to be displayed in a terminal should be sanitized to prevent escape sequences injection.

References

- https://i.blackhat.com/BH-US-23/Presentations/US-23-stok-weponizing-plain-text-ansi-escape-sequences-as-a-forensic-nightmare-appendix.pdf

4.3. Denial of Service via Event Flooding

Severity	MEDIUM
Affected Resources	include/gadget/buffer.h pkg/operators/ebpf/tracer.go
Status	Closed

Update

The vulnerability was reported through the GitHub Security section of the repository, in the linked advisory. It was fixed on release v0.50.1 and was assigned CVE-2026-31890.

Description

The `include/gadget/buffer.h` file contains definition of the Buffer API that gadgets can use to, among the other things, transfer data from eBPF programs to user-space.

For hosts running a modern enough Linux kernel (>= 5.8), this transfer mechanism is based on ring-buffers.

Gadgets have mainly two ways of transferring data via ring-buffers:

1. `gadget_reserve_buf` which ends up in a `gadget_discard_buf` or a `gadget_submit_buf`, where a pointer to the kernel ring-buffer is directly retrieved and can be used to write data.
2. `gadget_output_buf`, used when data is first built "locally" into the eBPF program and then copied over to the ring-buffer.

Gadgets declare the map, as per documentation, via:

```
GADGET_TRACER_MAP(events, 1024 * 256);
```

and then event tracing is implemented as follows:

```
static __always_inline int trace_exit(struct syscall_trace_exit *ctx)
{
    struct event *event;
    /* ... */
    event = gadget_reserve_buf(&events, sizeof(*event));
    if (!event)
        goto cleanup;

    /* fill the event here */
    /* ... */
    gadget_submit_buf(ctx, &events, event, sizeof(*event));
    /* ... */
}
```

In the Golang user-space, the flow is managed by `pkg/operators/ebpf/tracer.go`, where event samples are read from the eBPF ring-buffer, together with a count of lost samples:

```
// SNIP
switch t.mapType {
case ebpf.RingBuf:
    var rec ringbuf.Record
    readCb = func() ([]byte, uint64, error) {
        err := t.ringbufReader.ReadInto(&rec)
        return rec.RawSample, 0, err
    }

// SNIP

for {
    sample, lost, err := readCb()
```

From the flow described above, three facts can be noted:

1. The size of the ring-buffer for the gadgets is hard-coded to 256KB.
2. When a `gadget_reserve_buf` fails because of insufficient space, the gadget silently cleans up without producing an alert.
3. The `lost` count reported by the eBPF operator, when using ring-buffers is hardcoded to zero.

Because of this, in a situation where the ring-buffer of a gadget is incidentally or maliciously already full, the gadget will silently drop events.

Impact

An attacker able to generate events (e.g. due to a compromised container) could leverage this vulnerability to cause a Denial of Service, forcing the system to drop events coming from other containers (or the same container).

In case tracing is used for security purposes, this can be abused by an attacker to compromise the integrity of the logs.

Attack Complexity

In order to flood the ring-buffer, the attacker would need a way to arbitrarily produce events (e.g. a compromised container or process).

Due to the limited default size of the map, flooding the buffer is relatively easy to obtain.

Related Issues

N/A

Proof of Concept

1. Setup a Linux host and build/install `ig` version `0.48.0`

2. Run the attached `run_poc4.sh` on a terminal
3. Run `sudo ig run trace_open -c poc-flood-evasion | grep shadow` on another terminal
4. Press "Enter" on the terminal attached to `run_poc4.sh`
5. Observe that the first access to `/etc/shadow`, performed before flooding, is caught by the gadget
6. Observe that the second access to `/etc/shadow`, performed while events are flooding the ring-buffer, is silently dropped

Suggested Remediations

Collect dropped packets and periodically process them from user-space. Inform the user with an alert in case multiple packets are lost.

If possible, implement container-based rate-limiting to throttle events from noisy containers, ensuring fairness among containers.

References

N/A

5. Hardening

This section highlights recommendations to improve the security posture of the project.

5.1. TLS Optional on TCP Listeners

When the daemon starts a TCP listener without TLS flags, it only logs a warning and proceeds in plaintext. Coupled with the complexity of properly managing mTLS, this increases likelihood of deploying insecure TCP services that can be abused by potential attackers.

Inspektor Gadget should instead follow a secure default policy: when spawning a TCP listener, TLS flags should be required, and disabling them should require an explicit option such as `--tcp-insecure`.

5.2. Unverified Resource Download and Execution

In the CI/CD supply chain of Inspektor Gadget, some external dependencies are downloaded without proper verification:

- `actionlint`: in the `inspektor-gadget.yml` workflow, the `download-actionlint.bash` script is fetched from the `main` branch of the `rhysd/actionlint` repository and executed. The script downloads the `actionlint` binary from a GitHub release. Neither stage performs any hash or signature verification.
- `vimto`: in the `inspektor-gadget.yml` workflow, the `vimto` package is downloaded using `@latest` instead of a pinned version or commit hash.
- `bpftool`: in the `tools/getbpftool.sh` script, the `bpftool` is resolved from `latest` dynamically from the GitHub API, without pinning its version.
- `actions/checkout`: in the `check-artifacthub-tags.yml` and `check-links.yml` workflows, the `actions/checkout` dependency is pinned to a tag – which can be forged – rather than a commit hash.

It is recommended, where feasible, to always pin the dependency to a specific hash/version, and correctly verify the signature of the external dependency.

5.3. Implement a Kubernetes Namespace Blocklist

Inspektor Gadget operators can filter gadget executions on Kubernetes-specific entities such as pods and namespaces, as described in the docs.

However, for single-node clusters, where a node can both be a control node and worker node, this can lead to unintended tracing on sensitive namespaces, such as `kube-system`.

Inspektor Gadget administrators should be able to configure a block-list of namespaces that cannot be filtered, to prevent leaks of sensitive cluster data rather than application workload data.

5.4. Prevent Clients from Enabling Host Tracing

On Linux standalone hosts (with an `ig daemon` running), clients that connect remotely via `gadgetctl` can run gadgets as follows to enable full-host tracing:

```
gadgetctl run trace_exec --host
```

Official gadgets such as `trace_exec`, `ttysnoop` or `tcdump` could be abused by malicious clients to compromise the node.

It is recommended to block clients from enabling host-level tracing, or at least to document the attached risk in the documentation of `ig daemon`, so to inform system administrators of the attached risks.

This hardening was initially submitted as a vulnerability through the GitHub Security section, but it was closed, as this behavior is allowed in the threat model of the project.

5.5. Automate Third Party Vulnerability Scanning

Inspektor Gadget pins a known vulnerable `golang` version on some of the project tools. It is recommended to add a vulnerability scanner such as the Google osv-scanner to automate listing known vulnerable dependencies, and mitigate whenever possible.

5.6. Dangerous GET nodes/proxy Permission

The RBAC role assigned to the DaemonSet pod where Inspektor Gadget is deployed on Kubernetes owns the following permission:

```
- apiGroups: [""]
  resources: ["nodes/proxy"]
  verbs: ["get"]
```

In 2026, this permission was reported to allow code execution in the cluster.

It's important to mention that, in order to leverage this permission, a potential attacker would require a highly privileged position, e.g. code execution in the `gadget` pod, which likely provides other escalation primitives, as it runs with the `CAP_SYS_ADMIN` Linux capability.

Nevertheless, considering that the `serviceaccount` token might be leaked in other ways, it is still recommended to follow the *Principle of Least Privilege* and restrict the role assigned to the DaemonSet.

6. Gadget bypasses

As aforementioned, one of the agreed objectives of this audit was to evaluate whether a compromised container could bypass the official gadgets, essentially executing the operations which are supposed to be traced by gadgets, without triggering any event. This section contains the result of this part of the audit.

It is recommended to harden the gadgets, for bypasses that have a reasonable fix (e.g. hook a previously ignored `syscall`) and to document limitations for bypasses that are intrinsic to the design of the eBPF code, as it happens, for example, with `uprobe`-based gadgets.

6.1. *trace_open Bypass via openat2*

Description

The `trace_open` only hooks `sys_enter_open` and `sys_enter_openat`, but not `sys_enter_openat2`, introduced in Linux 5.6.

Proof Of Concept

1. Execute the `run.sh` script inside the `poc-openat2` attached directory and follow the instructions
2. Notice that Inspektor Gadget does not log the file read marked as `[INVISIBLE]` in the code

```
ig-audit:workdir$ sudo ig run trace_open -c poc-openat2 | grep OPENAT
WARN[0000] This gadget was built with ig v0.48.1 and it's being run with v0.48.0. Gadget could be incompatible
WARN[0000] This gadget was built with ig v0.48.1 and it's being run with v0.48.0. Gadget could be incompatible
poc-openat2                    openat2_poc                994898          994898 994862                          3 /tmp/VISIBLE_OPENAT
                                          -rw-r--r--
poc-openat2                    openat2_poc                994898          994898 994862                          3 /tmp/VISIBLE_OPENAT_CONTROL
                                          -rw-r--r--
```

6.2. *trace_mount Bypass via fs***

Description

The `trace_mount` gadget only hooks `mount()` and `umount()` syscalls, but not the `(fsopen/fsconfig/fsmount/move_mount)` mount API introduced in Linux 5.2.

Proof Of Concept

3. Execute the `run.sh` script inside the `poc-fsmount` attached directory and follow the instructions
4. Notice that Inspektor Gadget does not log the operation marked as `[INVISIBLE]` in the code

```
ig-audit:workdir$ sudo ig run trace_mount -c poc-new-mount
WARN[0000] This gadget was built with ig v0.48.1 and it's being run with v0.48.0. Gadget could be incompatible
WARN[0000] This gadget was built with ig v0.48.1 and it's being run with v0.48.0. Gadget could be incompatible
RUNTIME.CONTAINERNAME        COMM                  PID        TID TID        FLAGS          CALL                          ERROR                DELTA
poc-new-mount                fsmount_poc        1003693    1003693 1003655                mount("tmpfs", "/mnt/poc_old…                      66.08µs
poc-new-mount                fsmount_poc        1003693    1003693 1003655                umount("/mnt/poc_old_api", "…                     189.50µs
poc-new-mount                fsmount_poc        1003693    1003693 1003655                umount("/mnt/poc_new_api", "…                      87.83µs
```

6.3. tcpdump Bypass via Jumbo Frames

Description

The `tcpdump` gadget truncates the captured packet at the value of the `snaplen` variable, which by defaults is equal to 1500 bytes (Ethernet MTU). However, if jumbo frames with a higher size are transmitted, this will lead to a truncated capture.

Proof of Concept

1. Execute the `run.sh` script inside the `poc-tcpdump` attached directory
2. Notice, from the script output, that the communication between the sender and receiver was not correctly captured by the `tcpdump` gadget.

6.4. trace_sni Bypass via IPv6

Description

The `trace_sni` gadget drops any packet that is not strictly IPv4. For this reason, TLS connections happening over IPv6 will evade SNI tracing.

Proof of Concept

1. Execute the `run.sh` script inside the `poc-sni` attached directory, following the instructions
2. Notice that Inspektor Gadget only logs the IPv4 TLS connection, while IPv6 goes through silently

```
ig-audit:workdir$ sudo ig run trace_sni -c poc-ipv6-client
WARN[0000] This gadget was built with ig v0.49.0 and it's being run with v0.48.0. Gadget could be incompatible
WARN[0000] This gadget was built with ig v0.49.0 and it's being run with v0.48.0. Gadget could be incompatible
RUNTIME.CONTAINERNAME                    COMM                  PID        TID TID        NAME
poc-ipv6-client                          curl              1051809    1051809 1051785    ipv4.example.com
```

6.5. Evasion of uprobe-based Gadgets

Description

Some gadgets, such as `trace_ssl` or `trace_malloc`, implement tracing on dynamically linked libraries, such as `libssl.so`. This design can be bypassed by either statically linking the target libraries, or implementing the library functionality by directly invoking syscalls.

Proof of Concept

1. Execute the `run.sh` script inside the `poc-uprobe` attached directory, following the instructions
2. Notice that Inspektor Gadget only logs the SSL operations of `curl` (that dynamically links OpenSSL), whereas operations coming from the golang binary are not traced, since the `crypto/tls` library is statically linked

```
ig-audit:workdir$ sudo ig run trace_ssl -c poc-uprobe-evasion
WARN[0000] This gadget was built with ig v0.49.0 and it's being run with v0.48.0. Gadget could be incompatible
WARN[0000] This gadget was built with ig v0.49.0 and it's being run with v0.48.0. Gadget could be incompatible
RUNTIME.CONTAINERNAME      COMM                PID         TID TID        BUF                    OPERATION     ERROR                  LATENCY_NS
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_d… error#4294967295          5.34ms
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_w… error#4294967195         59.54µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_r… error#4294967256        149.17µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_w… error#4294967287         46.92µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_r… error#4294967287         52.92µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_r… error#4294967160         48.83µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_r… error#4294966760         12.42µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_r… error#4294967286          6.71µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_r… error#4294967287         50.42µs
poc-uprobe-evasion         curl            1073934     1073934 1073664    <8192 bytes>           libssl_SSL_w… error#4294967270         71.42µs
```

6.6. Evasion via io_uring

Description

Many gadgets implement tracing by hooking syscall entry/exit tracepoints. However, as detailed in the linked ARMO security research, `io_uring` allows an alternative paths to syscall that would evade the syscall-based tracing offered by Inspektor Gadget.

However, it's worth mentioning that, in order to use `io_uring`, a container needs to run with an `unconfined` seccomp profiles, which is not the default in Docker.

Proof of Concept

1. Execute the `run.sh` script inside the `poc-uring` attached directory, following the instructions
2. Notice that Inspektor Gadget does not log the `openat` syscall invoked via `io_uring`

```
ig-audit:workdir$ sudo ig run trace_open -c poc-io-uring | grep OPENAT
WARN[0000] This gadget was built with ig v0.48.1 and it's being run with v0.48.0. Gadget could be incompatible
WARN[0000] This gadget was built with ig v0.48.1 and it's being run with v0.48.0. Gadget could be incompatible
poc-io-uring               io_uring_open           1082862         1082862 1082822                    3 /tmp/VISIBLE_OPENAT
                                          -rw-r--r--
poc-io-uring               io_uring_open           1082862         1082862 1082822                    3 /tmp/VISIBLE_OPENAT_CONTROL
                                          -rw-r--r--
```

7. Attachments

The following is the list of attachments and the related SHA256 integrity hashes.

7.1. *finding2/run_poc2.sh*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/finding2/run_poc2.sh

SHA256: `648e73bc65380b28495844e5778a844c43b4f3bcfbaa3c32fd33807e3b4439ff`

7.2. *finding2/escape_inject.c*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/finding2/escape_inject.c
SHA256: `3847d79f7e109f8eb76cac1bbb018792104581f0da21e616c4198c6f5565e0f2`

7.3. *finding4/run_poc4.sh*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/finding4/run_poc4.sh
HA256: `f2d0a2573b4c680b8e3eb1b06225e7c8b9d2081b589b03c92b94b670ceb25cbf`

7.4. *finding2/evade_flood.c*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/finding4/evade_flood.c
SHA256: `abbacdbf8ecf1e1d1cad4bb803f575a2d354501ca022c746c42ab7e7be4ee70d`

7.5. *poc-openat2/openat2.c*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-openat2/openat2.c
SHA256: `52acc5ab4d9a5bef24c423c97439fe530114adfb64436c0d43ae1b7711c6ae30`

7.6. *poc-openat2/run.sh*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-openat2/run.sh
SHA256: `a4f87f5e8b01e020ec8a55533e3f3e041ce36969c2645c8823b9b57adfe8f2ea`

7.7. *poc-fsmount/fsmount.c*

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-fsmount/fsmount.c
SHA256: `8470dc71477e39c0408a9073394faf7f9eb1f2857f66f4a3ae97dde51b049b82`

7.8. poc-fsmount/run.sh

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-fsmount/run.sh
SHA256: `e6e2287869f02bab9ea723466579d364402c04d246e02e60f63a938a05b157f5`

7.9. poc-tcpdump/sender.py

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-tcpdump/sender.py
SHA256: `f85e45f3513425aaa67d2885148bef46fca807715dc039d512bcf8f892649e30`

7.10. poc-tcpdump/receiver.py

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-tcpdump/receiver.py
SHA256: `244a643841408a905683f5474862f5683fd8be36d0b4d1da0a43eb81443bf616`

7.11. poc-tcpdump/run.sh

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-tcpdump/run.sh
SHA256: `6e86f014ad2604af1f5199ea5e7b284f34b1c63e787d3c87e6013922a22be7a3`

7.12. poc-sni/server.py

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-sni/server.py
SHA256: `7c39b3a5af8fd8813f7969df43e2c033cd7f51acd0c299aadb9214a44d306a7d`

7.13. poc-sni/run.sh

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-sni/run.sh
SHA256: `61ec1b24e3e1069bf2f068d731bc6c95f035318b3f043165ecc3d627f57032f0`

7.14. poc-uprobe/static_tls.go

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-uprobe/static_tls.go
SHA256: `a0fb47b499ba95425e3f71b6488d7f1927e28590e5c1205ad9f82bfa53933aa0`

7.15. poc-uprobe/run.sh

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-uprobe/run.sh
SHA256: `71b1b2c0e602de39d79cabcbe6b3cb8904624867d6a5834b92c25faf48edfebd`

7.16. poc-uring/io_uring_open.c

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-uring/io_uring_open.c
SHA256: `8df62049138bc3ebdaf6b093cdd2cb5902a2ecb6e061b03b511e2666856add11`

7.17. poc-uring/run.sh

Link: https://github.com/ShielderSec/poc/blob/main/inspektor-gadget/poc-uring/run.sh
SHA256: `a63a2ce83c53b6ee26438bcfc80fad392d844d2e74ea71ab82da6cbeeccad211`