Edera Container Runtime

Security Assessment

October 21, 2025

Prepared for:
Jed Salazar
Edera, Inc.

Prepared by: **Spencer Michaels**

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Tara Goodwin-Ruffus Project Manager
tara.goodwin-ruffus@trailofbits.com@trailofbits.com

The following engineering director was associated with this project:

Keith Hoodlet, Engineering Director, AI/ML and Application Security
keith.hoodlet@trailofbits.com

The following consultant was associated with this project:

Spencer Michaels, Consultant
spencer.michaels@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
August 4, 2025	Pre-project kickoff call
September 22, 2025	Status update meeting #1
September 29, 2025	Status update meeting #2
October 3, 2025	Status update #3 (report provided, no meeting)
October 10, 2025	Delivery of report draft
October 16, 2025	Report readout meeting
October 21, 2025	Delivery of final comprehensive report

Executive Summary

Engagement Overview

Edera, Inc. engaged Trail of Bits to review the security of its hardened container runtime. Edera provides a hypervisor that isolates containers inside lightweight virtual machines (VMs), exposing a Kubernetes-compatible interface for managing said containers. The scope of the audit included the following components.

- The Edera Protect daemon, which is responsible for managing zone and workload life cycles and provides a control API to do so
- The Styrolite container runtime, which is responsible for sandboxing workloads via Linux kernel namespacing
- Edera's patches to the Xen hypervisor, which primarily involve changes to the `vpci` (virtual PCI) and `x86pv` (paravirtualization) drivers to support PCIe passthrough, changes that are necessary to enable passthrough of GPUs to Edera zones
- Rust reimplementations of the Xen kmod service and userspace components, intended to provide better security guarantees than the original implementations
- Build scripts used to produce OCI images with Edera's built-in software

One consultant conducted the review from September 15 to October 10, 2025, for a total of four engineer-weeks of effort. Our testing efforts focused on identifying vulnerabilities that could compromise the isolation boundaries between different zones or between a zone and the host. With full access to source code, documentation, and a live testing environment, we performed static and dynamic testing of the target codebase, using automated and manual processes.

Observations and Impact

The security posture of the Edera Protect daemon and its surrounding infrastructure is generally robust; we identified no medium- or high-severity findings during this audit. While we did not identify any vulnerabilities that would compromise the primary isolation guarantees of the system (chiefly, that zones are isolated from one another and from the host), we did note a significant number of lower-severity lapses in input validation and defense-in-depth measures. Such issues can facilitate lateral movement or privilege escalation by an attacker who has obtained a foothold through other means.

Nearly two-thirds of the issues identified in this report were the result of insufficient data validation. We observed insecure handling of path concatenation in multiple locations (TOB-EDRA-2 and TOB-EDRA-11), as well as a pattern of missing or incorrect validation measures for string-based inputs (TOB-EDRA-1, TOB-EDRA-3, TOB-EDRA-6, TOB-EDRA-12).

Default-permissioned temporary files are used to pass workload configurations to the Styrolite binary in several locations, needlessly exposing those configurations to threats from an attacker with low-privileged filesystem access (TOB-EDRA-4, TOB-EDRA-15). Creating and permissioning a directory in two steps, as noted in TOB-EDRA-7, could enable similar filesystem-based attacks.

Finally, we noted several cases in which callers failed to check the return value of certain memory management functions, or to validate pointers, sizes, and array indices before acting on them (TOB-EDRA-8, TOB-EDRA-9, TOB-EDRA-10, TOB-EDRA-13, TOB-EDRA-14). In exceptional cases, albeit none of which we identified as attacker-triggerable, this could result in zones being left in an inconsistent state, with the low-level memory map diverging from a safe state assumed by higher-level code.

In summary, however, none of the issues described above directly allow an attacker to break out of a zone and onto the host or move laterally between zones. Many of these issues can be mitigated going forward by stricter application of secure development practices for specific operations (such as establishing known-safe standards for handling temp files, creating directories, etc.) and enforcing them with static analysis rules that flag potential lapses in security hygiene.

Recommendations

Based on the findings identified during the security review, Trail of Bits recommends that Edera take the following steps:

- **Remediate the findings disclosed in this report**, prioritizing those affecting data validation and handling. These findings should be addressed through direct fixes or broader refactoring efforts.

- **Identify sources and sinks for user input and comprehensively establish appropriate validation measures.** Especially when receiving arbitrary text input, such as configuration strings, ensure that only a known-good set of values is accepted.

- **Eliminate the use of temporary files for passing data between applications.** Ideally, pass Styrolite container configurations in-memory without touching the filesystem.

- **Explicitly check arithmetic operations, especially when constructing pointers or indexing into arrays.** Do not rely on implicit assumptions that, for instance, a function's return value will always be within a safe range for a later array index operation. Implement such operations so that they are inherently safe rather than relying on implicit behavior.

- **Ensure that return codes are checked, especially for unsafe calls to C library functions.** Failure to check return status can leave the application in an inconsistent state on a failure.

- **Represent paths using Rust's native `std::path` type.** Incoming path strings should be converted into `std::paths` before manipulating them, and `std::path::join` should be used for all path concatenation.

- **Consider employing static analysis using tools such as Semgrep and CodeQL to better identify the above issues.** For instance, ignored return codes can be automatically detected by a linter triggered by a commit hook.

- **Consider conducting a targeted review of other portions of the Xen codebase that present particular concern for Edera.**

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	0
Low	10
Informational	5
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Exposure	2
Data Validation	9
Timing	1
Undefined Behavior	3

Project Goals

The engagement was scoped to provide a security assessment of the Edera container runtime. Specifically, we sought to answer the following non-exhaustive list of questions:

- Can an attacker with privileged access inside a zone break out to gain access to either the host or another zone?
- Are there vulnerabilities in the Xen hypervisor, especially its paravirtualized guest drivers, that could allow an attacker to break isolation between the zone and the host, or between zones?
- Do any of the modifications introduced by Edera's patches (primarily in the `vpci` and `x86pv` drivers) alter Xen's behavior in ways that would leak guest memory, especially when using GPU passthrough?
- When launching new zones, are the guest's memory mappings and page grants initialized correctly?
- Does the Protect daemon launch zones in a configuration that is secure by default? Can zones be configured in ways that could give them unexpected access to other zones' data?
- Are the containers running within zones resilient to breakouts? While the underlying VM provides the ultimate security boundary, well-secured containers offer a line of defense in depth.

Project Targets

The engagement involved reviewing and testing the following targets.

Xen (`edera/4.21` branch)

Repository	https://github.com/edera-dev/xen/compare/master...edera/4.21
Version	`2c00127e8bca4890461825fc7a65e830c98e34c1`

Edera Protect daemon

Repository	https://github.com/edera-dev/protect/
Version	`274777592425ca79382faa4b165a675a379c9363`

Styrolite

Repository	https://github.com/edera-dev/styrolite/
Version	`6a267ec0e0fa7df728298160e57827bcc9680d98`

rkmod

Repository	https://github.com/edera-dev/rkmod/
Version	`9b59f7b455b2a10a52a219524b6f07d15b26ff3d`

Linux kernel OCI build scripts

Repository	https://github.com/edera-dev/linux-kernel-oci/
Version	`cda447994f828506173741d09272d0c6f1a5c223`

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manual code review of the Edera Protect, Styrolite, `rkmod`, and `linux-kernel-oci` codebases, as well as portions of the Xen codebase, with a focus on components affected by patches made to Xen by Edera's developers
    - Review of Edera's reimplementations of the Xen libraries `xencall`, `xenclient`, `xenevtchn`, `xengnt`, `xenplatform`, and `xenstore`, which are used by the Protect daemon
- Static analysis of the in-scope code noted above via Semgrep and CodeQL
- Dynamic testing of the Edera Protect daemon in a live environment

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- Because the Xen hypervisor comprises an extremely large volume of third-party code, and because of the limited time available for the audit, we restricted our audit of third-party code to the portions of the Xen codebase that were closely related to the changes made to Xen by Edera, with a focus on GPU passthrough and page table allocation.

Summary of Findings

The table below summarizes the findings of the review, including details on type and severity.

ID	Title	Type	Severity
1	CreateRequest and AttachRequest validation is bypassed	Data Validation	Informational
2	Styrolite can mount to directories outside a target container	Data Validation	Low
3	Resource limits can be used to set arbitrary cgroup keys	Data Validation	Low
4	Styrolite configuration needlessly passes through the filesystem	Data Exposure	Low
5	SSRF vulnerability in OCI image authentication	Data Validation	Low
6	OCI connects to Docker hub mirrors starting with “localhost” using HTTP	Data Validation	Informational
7	Two-step directory creation vulnerable to race condition	Timing	Low
8	Missing call to destroy_map_task	Undefined Behavior	Informational
9	Unchecked return values during grant unmapping	Undefined Behavior	Low
10	map_vf can fail silently	Undefined Behavior	Informational
11	Unsanitized string-wise mount path concatenation in zone crate	Data Validation	Low

12	is_edera_runtime_class improperly identifies the runtime class	Data Validation	**Informational**
13	Excessive (4 GB) memory consumption for IDM packets	Data Validation	**Low**
14	Page number overflow can cause driver crash at zone boot	Data Validation	**Low**
15	Workload configuration written to temp file	Data Exposure	**Low**

Detailed Findings

1. CreateRequest and AttachRequest validation is bypassed	
Severity: **Informational**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-EDRA-1
Target: `styrolite/src/config.rs:185–195`	

Description

Users of Styrolite can submit either a `CreateRequest` or `AttachRequest` to create a container or attach to an existing one, supplying a set of parameters for the action in either case. These requests are intended to be validated via the `Validatable` trait, and, indeed, both implement it; however, the implementations are stubbed out and unconditionally accept any submitted configuration.

```
pub trait Validatable {
    /// Validate the configuration and error if the configuration is invalid.
    fn validate(&self) -> Result<()>;
}

impl Validatable for CreateRequest {
    fn validate(&self) -> Result<()> {
        Ok(())
    }
}

impl Validatable for AttachRequest {
    fn validate(&self) -> Result<()> {
        Ok(())
    }
}
```

Figure 1.1: The `validate` function unconditionally returns `Ok(())` for `CreateRequest` and `AttachRequest`. (`styrolite/src/config.rs:180–195`)

Recommendations

Short term, fully implement the two `validate` functions noted above.

Long term, if it is necessary to stub out measures that will be implemented later, especially ones used for validation or security-adjacent operations, flag them in code and/or at runtime, such as with a “FIXME” comment and a debug message.

2. Styrolite can mount to directories outside a target container

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EDRA-2
Target: `styrolite/src/wrap.rs:284`	

Description

When launching a new container, Styrolite pivots the new namespace's filesystem root into a new root directory and then mounts any additional mount targets (such as `/proc`) below that new root. Incoming container configurations can include arbitrary mount targets, which are mounted like so:

```
if let Some(mounts) = &self.mounts {
    for mount in mounts {
        let parented_target = format!("{}/{}", rootfs, mount.target);
        let parented_mount = MountSpec {
            source: mount.source.clone(),
            target: parented_target.clone(),
            fstype: mount.fstype.clone(),
            bind: mount.bind,
            recurse: mount.recurse,
            unshare: mount.unshare,
            safe: mount.safe,
            create_mountpoint: mount.create_mountpoint,
        };

        parented_mount
            .mount()
            .expect("failed to process mount spec");
    }
}
```

Figure 2.1: Mounting of optional targets (`styrolite/src/wrap.rs:282–300`)

Due to the string-based path join on line 284 (highlighted in figure 2.1), mount targets that contain references to the parent directory (i.e., `../`) can cause the `parented_target` to point to directories outside of `rootfs` (the directory that will become the container's root filesystem).

The client noted that configurations submitted to Styrolite come from the Edera Protect daemon and are considered trusted; therefore, this issue is of low severity in practice.

Exploit Scenario

A malicious container configuration could specify a target that mounts over a directory on the host machine, potentially allowing for the replacement of security-relevant files (e.g., root's SSH keys, configurations in `/etc`, and so on). In both of these cases, however, the "host" in question is merely the VM underlying a single container.

Similarly, a mount mapping could specify as its source a sensitive directory on the host to be mounted into the container, potentially exfiltrating data.

Recommendations

Short term, modify the above-noted code to first canonicalize `mount.target` and ensure that the result is a subdirectory of `rootfs` before proceeding to mount the directory.

Long term, also consider implementing validation of the `mount.source` value, limiting which host directories can be mounted into the container.

3. Resource limits can be used to set arbitrary cgroup keys

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EDRA-3
Target: `styrolite/src/wrap.rs:205–218`, `styrolite/src/runner.rs:203–213`	

Description

Container configurations submitted to Styrolite can contain a set of key-value pairs of cgroup settings intended to limit the target container's resources. The resource limits are added via `push_resource_limit`, which inserts arbitrary strings as key-value mappings:

```
pub fn push_resource_limit(mut self, key: &str, value: &str) -> CreateRequestBuilder
{
    if self.config.limits.is_none() {
        self.config.limits = BTreeMap::new().into();
    }
    if let Some(ref mut map) = self.config.limits {
        map.insert(key.to_string(), value.to_string());
    }
    self
}
```

Figure 3.1: Addition of resource limits (`styrolite/src/runner.rs:203–213`)

The limits are later written to `/sys/fs/cgroup/<container-id>/` as follows. It is possible to write not just limits, but also arbitrary configuration values for the cgroup:

```
let limits = self.limits.clone().unwrap();
let _: Vec<_> = limits
    .into_iter()
    .map(|(k, v)| {
        debug!("configuring resource limit {k} = {v}");
        match subtree.clone().set_child_value(&k, &v) {
            Ok(_) => (),
            Err(e) => {
                warn!("unable to set resource limit '{k}': {e:?}");
            }
        }
    }).collect();
```

Figure 3.2: Arbitrary key-value writes to a cgroup configuration (`styrolite/src/wrap.rs:205–218`)

Exploit Scenario

A container configuration is submitted with “limits” containing `cgroup.procs` (which can be used to move a process into the croup), `cgroup.subtree_control` (which can be used to enable additional controllers within the cgroup), or another sensitive configuration value.

Recommendations

Short term, modify `push_resource_limit` to use a whitelist of known-safe keys used for configuring resource limits only and to reject all other keys.

Long term, when interpolating arbitrary inputs into a configuration, ensure that only known-safe values can be submitted.

4. Styrolite configuration needlessly passes through the filesystem

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-EDRA-4
Target: `styrolite/src/runner.rs:364–368`	

Description

Styrolite's `run` function takes a container configuration as a parameter and launches a container accordingly. Internally, it first writes the configuration to a temp file; it then launches a new Styrolite process, passing in the temp file's path.

```rust
/// Run the specified container.
/// Returns exit code on success, else error.
pub fn run<T: Configurable>(&self, config: T) -> Result<i32> {
    let mut config_file = TempFile::new("litewrap-cfg-", ".json")?;
    self.write_config(config, &mut config_file)?;

    let status = self.create_command(&config_file)?.status()?;
    if let Some(code) = status.code() {
        return Ok(code);
    }

    Err(anyhow!("failed to launch/monitor child process"))
}
```

Figure 4.1: Writing the container configuration to an intermediate temporary file (`styrolite/src/runner.rs:362–374`)

This exposes the configuration to a race condition, as in many default system configurations the temp file will be written to `/tmp` with world-writable permissions.

Exploit Scenario

An unprivileged attacker on the same system as Styrolite watches the `/tmp` directory for newly created configuration files and overwrites one as soon as it appears. By the time the new Styrolite binary launches and loads the configuration, it will process the modified configuration.

Recommendations

Short term, modify Styrolite's runner to receive a configuration on `stdin` if no file path is specified and change callers to pipe the configuration in when launching the new binary. This will remove the need for writing the configuration to the filesystem in the above case.

```rust
use std::process::{Command, Stdio};
use std::io::Write;

let mut child = Command::new("<COMMAND>")
    .stdin(Stdio::piped())
    .spawn()
    .expect("Failed to spawn");

let stdin = child.stdin.as_mut().expect("Failed to open stdin");
stdin.write_all(b"<CONFIGURATION>").expect("Failed to write");

// Optionally close stdin to signal EOF
drop(stdin);

let output = child.wait_with_output().expect("Failed to read output");
```

Figure 4.2: An example of piping the configuration directly to `stdin` for a new process

Long term, avoid writing intermediate work products of a process to disk whenever possible. If doing so is necessary, use a temp file with the strictest possible read/write permissions, and delete it as soon as it is no longer needed.

5. SSRF vulnerability in OCI image authentication

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EDRA-5
Target: `protect/crates/oci/src/registry.rs:169–212`	

Description

When the `oci` crate sends a request to a registry and authentication fails, it can reattempt the request based on the `www-authenticate` header returned by the registry, ultimately sending an HTTP GET request to the URL specified in the `realm` value of that header, including in cases where that value specifies an internal hostname or RFC 1918 address.

```
async fn call_once(&mut self, req: RequestBuilder) -> Result<Option<Response>> {
        let req_first_try = req.try_clone().ok_or(anyhow!("request is not 
clonable"))?;
        let req_first_try = self.apply_auth(req_first_try);
        let response = self
            .context
            .http
            .agent()
            .execute(req_first_try.build()?)
            .await?;
        if response.status() == StatusCode::UNAUTHORIZED {
            debug!("got unauthorized response, attempting to fetch/refresh token");
            let Some(www_authenticate) = response.headers().get("www-authenticate") 
else {
                bail!("not authorized to perform this action");
            };

            let www_authenticate = www_authenticate.to_str()?;
            if !www_authenticate.starts_with("Bearer ") {
                bail!("unknown authentication scheme");
            }

            let details = &www_authenticate[7..];
            let details = details
                .split(',')
                .map(|x| x.split('='))
                .map(|mut x| (x.next(), x.next()))
                .filter(|(key, value)| key.is_some() && value.is_some())
                .map(|(key, value)| {
                    (
                        key.unwrap().trim().to_lowercase(),
                        value.unwrap().trim().to_string(),
                    )
                })
                .map(|(key, value)| (key, value.trim_matches('\"').to_string()))
                .collect::<HashMap<_, _>>();
            let realm = details.get("realm");
            let service = details.get("service");
            let scope = details.get("scope");
            if realm.is_none() {
                bail!("unknown authentication scheme: realm required");
            }
            let mut url = Url::parse(realm.unwrap())?;

            if let Some(service) = service {
                url.query_pairs_mut().append_pair("service", service);
            }

            if let Some(scope) = scope {
                url.query_pairs_mut().append_pair("scope", scope);
            }

            let token_req = self.context.http.agent().get(url.clone());
            let token_req = self.apply_auth(token_req);
            let token_response = token_req.send().await?;
```

Figure 5.1: SSRF vulnerability in `protect/crates/oci/src/registry.rs:169–212`

Exploit Scenario

An attacker compromises a registry and intentionally returns a 401 Unauthorized response to an incoming request from the `oci` crate, providing a `www-authenticate` header with a realm specifying a hostname or address internal to the network on which the Protect daemon resides. The service then issues the GET request to that internal service.

Recommendations

Short term, before the second GET request is sent on line 210, have the `oci` crate resolve the hostname in the URL and determine whether it resolves to an RFC 1918 address. If so, it should not issue the request.

Long term, if making requests to URLs based on arbitrary input, always include validation to ensure that the URLs point to the external internet only.

6. OCI connects to Docker hub mirrors starting with “localhost” using HTTP

Severity: **Informational**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-EDRA-6
Target: `protect/crates/oci/src/name.rs:148–152`	

Description

The `oci` crate’s `ImageName` type contains a `registry_url` function that returns a `Url` object pointing to a target registry. It will use the `http://` scheme if the specified registry’s hostname *starts with* the string “localhost”; otherwise, it will use `https://`.

```
let url = if self.hostname.starts_with("localhost") {
    format!("http://{hostname}")
} else {
    format!("https://{hostname}")
};
```

Figure 6.1: Conditionally setting the URL scheme based on a “localhost” prefix (`protect/crates/oci/src/name.rs:148–152`)

The intention is likely to avoid needing to set up TLS certificates for a local registry used during testing; however, any registry with a hostname beginning with the string “localhost” will be connected to with the plaintext `http://` scheme. Since such a situation is vanishingly unlikely, this issue is of informational severity.

Recommendations

Short term, change the above-noted code to check whether the hostname *equals* “localhost”, rather than whether it begins with that string.

Long term, be especially cautious about using string prefix checks when analyzing hostnames—these are a common source of error when used carelessly.

7. Two-step directory creation vulnerable to race condition

Severity: **Low**	Difficulty: **High**
Type: Timing	Finding ID: TOB-EDRA-7
Target: `protect/crates/zone/src/init/early/util.rs:14–24`	

Description

The `create_dir` helper function in the zone crate takes a target path and a permissions mask (mode) and creates a directory with those permissions. However, it does so in two discrete steps, creating the directory on the filesystem first and then setting its permissions thereafter.

```
pub fn create_dir(path: &str, mode: Option<u32>) -> Result<()> {
    let path = Path::new(path);
    if !path.is_dir() {
        fs::create_dir(path)?;
    }
    if let Some(mode) = mode {
        let permissions = Permissions::from_mode(mode);
        fs::set_permissions(path, permissions)?;
    }
    Ok(())
}
```

Figure 7.1: Creating a directory with default permissions first and then setting permissions afterward (`protect/crates/zone/src/init/early/util.rs:14–24`)

Exploit Scenario

An attacker watches a parent directory that the Protect daemon will create a directory in, and he has write access to that parent directory. As soon as Protect creates a new directory, the attacker replaces it with a symlink to another directory on the system. Protect’s `set_permissions` call will then be directed at the symlink’s target. This could allow an attacker to change permissions on a file that the Protect daemon itself has access to, even if he does not have such access.

Recommendations

Short term, use DirBuilder to create a directory that has the desired permissions at the time of creation (e.g., using `DirBuilder::new().create(path).mode(mode)?`).

Long term, always atomically set permissions on resources at the time of creation; do not create and set permissions in separate steps.

8. Missing call to destroy_map_task

Severity: **Informational**	Difficulty: **N/A**
Type: Undefined Behavior	Finding ID: TOB-EDRA-8
Target: `xen/xen/drivers/vpci/header.c:462–471`	

Description

The function `vpci_modify_bars`, used to map base registers for virtual PCI devices, begins by allocating a `vpci_map_task`, which must be deallocated (via `destroy_map_task`) when no longer in use. The `destroy_map_task` function is called before return statements in the event of unrecoverable errors, except in the case of a failed call to `pci_sanitize_bar_memory`; this appears to be in error, with the original call having been removed during a change to the surrounding code in commit `47581ef7`. In the event that BAR memory sanitization fails during a call to `vpci_modify_bars` on Dom0, a resource leak will occur.

```
int vpci_modify_bars(const struct pci_dev *pdev, uint16_t cmd,
                     enum vpci_map_op map_op, bool rom_only)
{
    struct vpci_header *header = &pdev->vpci->header;
    struct pci_dev *tmp;
    const struct domain *d;
    const struct vpci_msix *msix = pdev->vpci->msix;
    struct vpci_map_task *task = alloc_map_task(pdev, cmd, map_op, rom_only);
    unsigned int i, j;
    int rc;

    // ... omitted for brevity ...

    if (is_hardware_domain(pdev->domain)) {
        rc = pci_sanitize_bar_memory(mem);
        if ( rc )
        {
            gprintk(XENLOG_WARNING,
                    "%pp: failed to sanitize BAR#%u memory: %d\n",
                    &pdev->sbdf, i, rc);
            // `task` is not deallocated before returning
            return rc;
        }
    }

    // ... omitted for brevity ...

    defer_map(task);
    return 0;
}
```

Figure 8.1: Relevant excerpts from `xen/xen/drivers/vpci/header.c:365–593`, showing the code block with the missing call to `destroy_map_task(task)`

Recommendations

Short term, add a call to `destroy_map_task(task)` just before `return rc` in the highlighted code block in figure 8.1.

Long term, consider using memory sanitization tooling such as Valgrind to identify resource leaks such as these.

9. Unchecked return values during grant unmapping

Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-EDRA-9
Target: `protect/crates/xen/xengnt/src/lib.rs:236–241`	

Description

The `xengnt` crate manages Xen's grant tables to permit domains to share memory pages. Granted pages can be unmapped using the `GrantTab::unmap` function, which is also called automatically via the `Drop` trait implemented on the `MappedMemory` datatype.

```rust
impl Drop for MappedMemory<'_> {
    fn drop(&mut self) {
        let _ = self.gnttab.unmap(self);
    }
}

impl GrantTab {

    // ... omitted for brevity ...

    fn unmap(&self, memory: &MappedMemory<'_>) -> Result<()> {
        let (offset, count) = self.device.get_offset_for_vaddr(memory.addr)?;
        let _ = unsafe { munmap(memory.addr as *mut c_void, memory.length) };
        self.device.unmap_grant_ref(offset, count)?;
        Ok(())
    }
}
```

Figure 9.1: Ignoring the return value of `munmap` at *`protect/crates/xen/xengnt/src/lib.rs:238`*

In `unmap`, the return value of the internal call to `munmap` is unchecked. In the event that `munmap` fails but `GrantDevice::unmap_grant_ref` succeeds, `unmap` will nonetheless return `Ok(())`, indicating success. In this case, the memory for the granted pages will remain mapped, while the grant reference in `xengnt` is removed, leaving `xengnt`'s representation of the grant table in an inconsistent state.

Exploit Scenario

A call to `munmap` on a granted page fails, but the failure is ignored due to the issue noted above. When `xengnt` issues further grants, it may reuse pages that are still mapped by the original grantee, potentially resulting in cross-domain memory access.

We were unable to conclusively determine whether granted pages on which `munmap` fails might be accessible to other guests in the future if the pages are reused—it appears that there *may* be some cases in which this can occur if additional domains hold references to the granted page.

In either case, `xengnt` is currently used only to map shared memory in order to access a domain's console; therefore, the attack surface of this vulnerability, even if successfully exploited, is likely to be small.

Recommendations

Short term, modify `GrantTab::unmap` to check the return value of `munmap` (which returns `-1` on failure) and return an `Err` value if it fails.

Also consider whether additional measures are needed to ensure the grant table remains in a consistent state in the event that `munmap` succeeds but `GrantDevice::unmap_grant_ref` fails.

Long term, always check the return value of functions that indicate success/failure via their return value. Consider using a static analysis tool such as Semgrep to flag cases where `let _ = ...` appears.

10. map_vf can fail silently

Severity: **Informational**	Difficulty: **N/A**
Type: Undefined Behavior	Finding ID: TOB-EDRA-10
Target: `xen/xen/drivers/vpci/sriov.c:169–170`	

Description

The `control_write` function in Xen's VPCI driver handles writes to the SR-IOV control registers of PCI devices with virtualization support. If `PCI_SRIOV_CTRL_MSE` ("Memory Space Enable") is newly set to 1, `control_write` will call `vf_map` across all virtual PCI devices (Virtual Functions). However, while `map_vf` can fail (e.g., if its internal BAR allocations fail), its return value is not checked in `control_write`. This could leave the VPCI driver in an inconsistent state, leaving a virtual PCI device unmapped.

```
static void cf_check control_write(const struct pci_dev *pdev, unsigned int reg,
                                   uint32_t val, void *data)
{
    unsigned int sriov_pos = reg - PCI_SRIOV_CTRL;
    uint16_t control = pci_conf_read16(pdev->sbdf, reg);
    bool mem_enabled = control & PCI_SRIOV_CTRL_MSE;
    bool new_mem_enabled = val & PCI_SRIOV_CTRL_MSE;

    ASSERT(!pdev->info.is_virtfn);

    if ( new_mem_enabled != mem_enabled )
    {
        if ( new_mem_enabled )
        {
            struct pci_dev *vf_pdev;

            /* FIXME casting away const-ness to modify vf_rlen */
            size_vf_bars((struct pci_dev *)pdev, sriov_pos);

            list_for_each_entry(vf_pdev, &pdev->vf_list, vf_list)
                map_vf(vf_pdev, PCI_COMMAND_MEMORY, VPCI_MAP);
        }
        /* TODO: unmap vf */
    }

    pci_conf_write16(pdev->sbdf, reg, val);
}
```

Figure 10.1: The `control_write` function (`xen/xen/drivers/vpci/sriov.c:150–176`)

```
/**
 * list_for_each_entry - iterate over list of given type
 * @pos:    the type * to use as a loop cursor.
 * @head:   the head for your list.
 * @member: the name of the struct list_head within the struct.
 */
#define list_for_each_entry(pos, head, member)                          \
    for ((pos) = list_entry((head)->next, typeof(*(pos)), member);      \
         &(pos)->member != (head);                                      \
         (pos) = list_entry((pos)->member.next, typeof(*(pos)), member))
```

Figure 10.2: The macro `list_for_each_entry` (`xen/xen/include/xen/list.h:489–498`)

Recommendations

Short term, change the above-noted lines of `control_write` such that the return value of `map_vf` is checked. Also consider, in the event of an error partway through the `for` loop, how previously successful device mappings should be handled—that is, whether they should be left mapped, or whether they should be unmapped again in a “rollback” to maintain consistency if the full set of mappings cannot be established.

```
list_for_each_entry(vf_pdev, &pdev->vf_list, vf_list) {
    int rc;
    if ((rc = map_vf(vf_pdev, PCI_COMMAND_MEMORY, VPCI_MAP)))
        return rc;
}
```

Figure 10.3: Suggested changes to the `control_write` function

Long term, always check the return value of functions that indicate success/failure via their return value. Consider using a static analysis tool such as Semgrep to flag cases where return values are ignored.

11. Unsanitized string-wise mount path concatenation in zone crate

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EDRA-11
Target: `protect/crates/zone/src/workload/sandbox.rs:320–335`	

Description

In a manner analogous to the path injection issue in Styrolite noted in TOB-EDRA-2, workload mount paths in the zone crate are also constructed using string-wise concatenation, with no sanitization step.

```
for mount in mounts.iter() {
    let mut source = PathBuf::from(format!(
        "{}/{}",
        self.volumes_path.to_string_lossy(),
        mount.tag
    ));
    if !mount.host_file.is_empty() {
        source.push(&mount.host_file);
    }
    let target = PathBuf::from(format!(
        "{}{}",
        self.overlay_path.to_string_lossy(),
        mount.target_path
    ));
```

Figure 11.1: Naive string-wise concatenation of mount paths (`protect/crates/zone/src/workload/sandbox.rs:320–335`)

Exploit Scenario

This issue has similar security properties to TOB-EDRA-2: a trusted configuration file, if modified, could result in mounts that are intended to be placed inside a workload’s filesystem instead of being mounted into the zone’s filesystem. This would not compromise the security boundary of the zone itself.

Recommendations

Short term, canonicalize all mount targets and add a check to ensure that the result is a subdirectory of the destination path before the code proceeds to mount the directory.

Long term, consider using a static analysis rule to preemptively flag uses of `PathBuf::from(format!(“{}{}”), . . .)` or `PathBuf::from(format!(“{}/{}”), . . .)`. The solution to this issue is sufficiently formulaic that it could likely be applied via an autofix Semgrep rule.

12. is_edera_runtime_class improperly identifies the runtime class

Severity: **Informational**	Difficulty: **N/A**
Type: Data Validation	Finding ID: TOB-EDRA-12
Target: `protect/crates/cri/src/cri.rs:134`	

Description

The `cri` crate contains a function `is_edera_runtime_class`, which determines whether a pod is using the Edera runtime class. This function is used to determine whether an image should be pulled from the Edera or delegate back end in the `image_status` function.

The `is_edera_runtime_class` function makes this determination by checking whether the pod's last applied configuration contains the string `"runtimeClassName":"edera"`. This does not guarantee that the `runtimeClassName` key is actually set to "edera", only that such a string is present *anywhere* in the configuration.

```
fn is_edera_runtime_class(spec: &Option<ImageSpec>) -> Option<bool> {
    // This is ugly but it works. Kubelet doesn't seem support runtime handler
    // all that well yet so all we get is an image ref and the annotations from
    // the pod. One of those annotations is the last applied config which will
    // contain the runtime class if it's configured. We can pull out metadata
    // from that annotation to see whether this pod is supposed to be edera or
    // default.
    spec.as_ref()
        .map(|spec| &spec.annotations)
        .and_then(|annotations| {
            annotations
                .get("kubectl.kubernetes.io/last-applied-configuration")
                .cloned()
        })
        .map(|last_applied| last_applied.contains("\"runtimeClassName\":\"edera\""))
}
```

Figure 12.1: Use of `contains` to check the runtime class in `is_edera_runtime_class` (`protect/crates/cri/src/cri.rs:120–135`)

Recommendations

Short term, change the above-noted function to actually parse the configuration JSON and check the specific value of the `runtimeClassName` key.

Long term, avoid using `contains` to check for the presence of a specific string value in a specific location. Err on the side of the strictest possible interpretation of such checks.

13. Excessive (4 GB) memory consumption for IDM packets

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EDRA-13
Target: `protect/crates/idm/src/client/file.rs:67–103`, `protect/crates/daemon/src/idm.rs:142–167`	

Description

The `idm` crate implements an IDM client, which receives packets through the `IdmFileBackend::recv` function. Each time a packet is received, this function reads a four-byte `size` value from the packet header and loops back to the `read_more` label until the specified number of bytes is received. The maximum size is 4 GB.

```
async fn recv(file: &mut File, read_buffer: &mut BytesMut) ->
Result<Vec<IdmTransportPacket>> {
    let mut data = vec![0; IDM_BUFFER_SIZE];
    let mut first = true;
    let mut packets = Vec::new();
    'read_more: loop {
        if !first {
            if !packets.is_empty() {
                return Ok(packets);
            }
            let size = file.read(&mut data).await?;
            read_buffer.extend_from_slice(&data[0..size]);
        }
        first = false;
        loop {
            if read_buffer.len() < 6 {
                continue 'read_more;
            }

            let b1 = read_buffer[0];
            let b2 = read_buffer[1];

            if b1 != 0xff || b2 != 0xff {
                error!(
                    "idm buffer was dirty, discarding {} bytes",
                    read_buffer.len()
                );
                read_buffer.clear();
                continue 'read_more;
            }

            let size = (read_buffer[2] as u32
                | ((read_buffer[3] as u32) << 8)
                | ((read_buffer[4] as u32) << 16)
                | ((read_buffer[5] as u32) << 24)) as usize;
            let needed = size + 6;
            if read_buffer.len() < needed {
                continue 'read_more;
            }
```

Figure 13.1: An excerpt of the `IdmFileBackend::recv` function (`protect/crates/idm/src/client/file.rs:67–103`)

Another file in the crate declares an `IDM_PACKET_MAX_SIZE` value of 20 MB; however, it is not enforced upon receipt.

```
const IDM_PACKET_QUEUE_LEN: usize = 100;
const IDM_REQUEST_TIMEOUT_SECS: u64 = 60;
const IDM_SAFE_GRACE_PERIOD_SECS: u64 = 60;
const IDM_PACKET_MAX_SIZE: usize = 20 * 1024 * 1024;
```

Figure 13.2: The maximum IDM packet size declared at `idm/src/client/mod.rs:44`

Therefore, packets can be received (and memory used up) in a size far in excess of what is necessary for an IDM packet.

A similar `recv` function is present in the IDM daemon, but it returns rather than loops if the length of the read buffer is insufficient to meet the declared size.

```
// read the size from the buffer as a little endian u32
let size = (buffer[2] as u32
    | ((buffer[3] as u32) << 8)
    | ((buffer[4] as u32) << 16)
    | ((buffer[5] as u32) << 24)) as usize;
let needed = size + 6;
if buffer.len() < needed {
    return Ok(());
}
```

Figure 13.3: The check after decoding the size header (`IdmService::process_rx_packet` `protect/crates/daemon/src/idm.rs:142–167`)

Exploit Scenario

An attacker in control of the IDM daemon sends an IDM packet with a size header of 0xFFFFFFFF (4 GB), but does not actually send that much data.

Alternately, the size header is corrupted in transit due to an error. The `recv` function on the client loops indefinitely, holding up to 4 GB in memory and wasting resources.

Recommendations

Short term, alter the two above-noted `recv` functions to add a check that `size` does not exceed `IDM_PACKET_MAX_SIZE`, rejecting any oversized packets.

Long term, in cases where a sender can specify a size header (and especially in which buffer space is pre-allocated based on an incoming size value), ensure that a sensible upper bound is enforced.

14. Page number overflow can cause driver crash at zone boot

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-EDRA-14
Target: `protect/crates/xen/xenplatform/src/x86pv.rs:668–726`	

Description

The `setup_page_tables` function, which executes during the boot process of a domain, reads mappings from the new domain's physical-to-machine (p2m) table. This occurs inside safe Rust code, so an invalid index cannot lead to an out-of-bounds read; however, an out-of-bounds access would result in an unhandled error, likely crashing the `x86pv` driver as a whole and affecting other guests. The arithmetic preceding the noted array access does not clearly bound the value of `pfn` to the length of the `p2m` table.

```
async fn setup_page_tables(&mut self, domain: &mut BootDomain) -> Result<()> {
    let p2m_segment = self
        .p2m_segment
        .as_ref()
        .ok_or(Error::MemorySetupFailed("p2m_segment missing"))?;
    let p2m_guest = unsafe {
        slice::from_raw_parts_mut(
            p2m_segment.addr as *mut u64,
            domain.phys.p2m_size() as usize,
        )
    };
    copy(p2m_guest, &domain.phys.p2m);

    for l in (0usize..X86_PGTABLE_LEVELS as usize).rev() {
        for m1 in 0usize..self.table.mappings_count {
            let map1 = &self.table.mappings[m1];
            let from = map1.levels[l].from;
            let to = map1.levels[l].to;
            let pg_ptr = domain.phys.pfn_to_ptr(map1.levels[l].pfn, 0).await? as
*mut u64;
            for m2 in 0usize..self.table.mappings_count {
                let map2 = &self.table.mappings[m2];
                let lvl = if l > 0 {
                    &map2.levels[l - 1]
                } else {
                    &map2.area
                };

                if l > 0 && lvl.pgtables == 0 {
                    continue;
                }
        if lvl.from >= to || lvl.to <= from {
          continue;
        }

        let p_s = (std::cmp::max(from, lvl.from) - from)
          >> (X86_PAGE_SHIFT + l as u64 * X86_PGTABLE_LEVEL_SHIFT);
        let p_e = (std::cmp::min(to, lvl.to) - from)
          >> (X86_PAGE_SHIFT + l as u64 * X86_PGTABLE_LEVEL_SHIFT);
        let rhs = X86_PAGE_SHIFT as usize + l * X86_PGTABLE_LEVEL_SHIFT as
usize;
        let mut pfn = ((std::cmp::max(from, lvl.from) - lvl.from) >> rhs) +
lvl.pfn;

        debug!(
          "setup_page_tables lvl={l} map_1={m1} map_2={m2} pfn={pfn:#x}
p_s={p_s:#x} p_e={p_e:#x}"
        );

        let pg = unsafe { slice::from_raw_parts_mut(pg_ptr, (p_e + 1) as
usize) };
        for p in p_s..p_e + 1 {
          let prot = self.get_pg_prot(l, pfn);
          let pfn_paddr = domain.phys.p2m[pfn as usize] << X86_PAGE_SHIFT;
          let value = pfn_paddr | prot;
          pg[p as usize] = value;
          pfn += 1;
        }
      }
    }
  }
  Ok(())
}
```

Figure 14.1: Excerpts of `BootSetupPlatform::setup_page_tables` (`protect/crates/xen/xenplatform/src/x86pv.rs:668–726`)

Exploit Scenario

Given that these operations occur as a domain is booting, it does not appear possible for a malicious guest to trigger a failure during page table setup. However, it may be possible to structure the memory layout of a malicious guest *image* in a way that it causes an invalid access into the p2m table when booted.

Recommendations

Short term, modify the above-noted code to explicitly check that the value of `pfn` is less than `domain.phys.p2m.len()` and safely bail out if not, aborting the guest boot process.

Long term, do not rely on implicit behavior to ensure that array indices or other bounded values fall within valid ranges. Instead, explicitly bounds-check those values and implement measures to safely recover from cases where they fall out of bounds.

15. Workload configuration written to temp file

Severity: **Low**	Difficulty: **High**
Type: Data Exposure	Finding ID: TOB-EDRA-15
Target: `protect/crates/zone/src/workload/process.rs:305–319`	

Description

When the Protect daemon starts a new workload, it creates a new Styrolite configuration and writes it to a file in the system's temp directory before passing it to Styrolite to launch the workload.

A similar issue is also noted in TOB-EDRA-4.

```
pub async fn start(&mut self, workload: Option<LocalWorkload>) -> Result<()> {
    // ... initialization omitted for brevity ...

    let styrolite_config: Config;

    // ... styrolite configuration instantiated ...

    tokio::fs::write(&tmp_file, serde_json::to_vec(&styrolite_config)?).await?;
    let tmp_file = tmp_file.to_string_lossy().to_string();
    let mut maybe_pty = None;
    let child = if self.spec.tty {
        let (pty, pts) =
            pty_process::open().map_err(|e| anyhow!("unable to allocate pty: {}",
e))?;
        let size = self
            .spec
            .terminal_size
            .map(|x| Size::new(x.rows as u16, x.columns as u16))
            .unwrap_or_else(|| Size::new(24, 80));
        pty.resize(size)?;
        maybe_pty = Some(pty);
        let mut command = pty_process::Command::new(BINARY);
        command = command.arg(&tmp_file);
```

Figure 15.1: An excerpt of `LocalWorkloadProcess::start` in which Styrolite is launched by supplying a configuration from a temp file (`protect/crates/zone/src/workload/process.rs:52–319`)

Exploit Scenario

An attacker with filesystem access to the temp directory on the zone VM watches for changes to that directory and injects a malicious configuration in place of one just written to that location, before it is loaded by Styrolite. In combination with TOB-EDRA-2, this could allow the attacker to escalate privileges by using Styrolite as a confused deputy to mount over sensitive directories on the zone to which the attacker's actual user account would not normally have access.

Since zones run only a single workload and are not recycled between workloads, this route of attack is of limited use unless an attacker were able to break out of a Styrolite container into the underlying zone VM and then manually trigger the start of a new workload within that zone to escalate privileges.

Recommendations

Short term, modify how Styrolite receives configuration files as described in TOB-EDRA-4.

Long term, avoid passing data between applications by way of temp files; or, if doing so is necessary (e.g., for compatibility reasons), explicitly permission those files so that they are not broadly writable by other parties on the system.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category does not apply to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

B. Code Quality Findings

This appendix contains findings that do not have immediate or obvious security implications. However, addressing them may enhance the code's readability and may prevent the introduction of vulnerabilities in the future.

- In the event that a Styrolite container's configuration does not contain a mount namespace (preventing Styrolite from pivoting to a new filesystem root), Styrolite will issue a warning but will otherwise continue. This warning may not be noticed, especially for an unattended Styrolite instance used as part of an automated pipeline. Consider having Styrolite bail out with an error in this case, and requiring an explicit command-line flag or configuration value to be set in order to bypass the warning.

```
if target_ns.contains(&Namespace::Mount) {
    self.pivot_fs()?;
} else {
    warn!("mount namespace not present in requested namespaces, trying to work
anyway...");
    warn!("this is an insecure configuration!");
}
```

Figure B.1: An easily missed security warning in `Wrappable::wrap` (`styrolite/src/wrap.rs:404–409`)

- The `cri` crate's service health check, `is_healthy`, uses the `all` function to check for the `Status::Alive` status on all services. In the event that there are no members in `ALIVE`, the check will return true. This may not be desirable behavior.

```
// If the map of service:healthy has all values == Alive, res will be a simple
200
// response. Otherwise, we must set res to have a StatusCode of 503
(SERVICE_UNAVAILABLE)
// If health checks were more than binary, time-sensitive checks, maybe we
could use more
// finesse...
fn is_healthy() -> bool {
    ALIVE
        .get()
        .unwrap() // SAFETY: This is the first thing we init after logging
        .lock()
        .is_ok_and(|x| x.values().all(|&status| status == Status::Alive))
}
```

Figure B.2: `is_healthy` (`protect/crates/cri/src/health.rs:64–70`)

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review assessments, supporting client organizations in the technology, defense, blockchain, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, Uniswap, Solana, Ethereum Foundation, Linux Foundation, and Zoom.

To keep up with our latest news and announcements, please follow @trailofbits on X or LinkedIn and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Trail of Bits considers this report public information; it is licensed to Edera, Inc. under the terms of the project statement of work and has been made public at Edera, Inc.'s request. Material within this report may not be reproduced or distributed in part or in whole without Trail of Bits' express written permission.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through sources other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

Trail of Bits performed all activities associated with this project in accordance with a statement of work and an agreed-upon project plan.

Security assessment projects are time-boxed and often rely on information provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and security properties. These techniques augment our manual security review work, but each has its limitations. For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. A project's time and resource constraints also limit their use.