Ootbi Product Security Assessment

Object First

September 16, 2024

Version 2.1

Prepared for:
Alexey Podoselnik
Eric Schott

Prepared by:
Andrew Havard
Rennie deGraaf
Vishtasp Jokhi
William Enright
William Groesbeck

1 Executive Summary

Synopsis

In late 2023 and 2024, Object First engaged NCC Group to conduct a security assessment of their Ootbi physical appliance server. Intended for use with Veeam Backup & Replication (VBR) v12, Ootbi serves as an immutable backup solution, where backups of other storage systems can be stored on Ootbi with assurance that the backups cannot be overwritten or deleted. Object First aimed to ensure that data stored on the server cannot be modified or deleted except as permitted by its retention policies and that it contains no other uncontrolled security risks.

NCC Group performed 40 person-days of testing against version 1.4 of the application in December 2023. Object First addressed the issues identified at that time in version 1.5, after which NCC Group performed an additional 14 person-days of testing in June and July 2024 focused specifically on verifying that the previously-identified issues had been fixed and on reviewing the application's immutability properties. This report contains the results of both rounds of testing.

Object First supported the project by providing NCC Group with a physical server appliance, a virtual machine image of the appliance software, application source code, documentation, and access to application developers.

Scope

NCC Group's evaluation included:

- **Storage API**: Web services API used by VBR and other systems to create buckets and store objects, issue and manage storage API credentials, and manage storage API permissions.
- **Management User Interface and API**: Web-based administrative console and underlying web services API to setup buckets, configure policies, and generally manage the device on a day-to-day basis.
- **Object First Ootbi Server**: On-premises solution that enables companies to retain immutable backups of storage buckets. Note that attacks requiring physical access were considered out-of-scope for this engagement.

Key Findings

The assessment uncovered 20 flaws, with ratings ranging between Informational and Critical. The issues rated Critical and High were:

- **OS Command Injection Leading to Remote Code Execution** - Some API endpoints did not properly sanitize user input and were vulnerable to OS command injection leading to remote code execution as the root user.
- **Weak Default Credentials** - The default user `objectfirst` and local Influx database were configured to use a weak and guessable password.
- **Application Disregards Service Names During Authorization** - The application did not correctly compare the API action being invoked to actions listed in permission policies, potentially allowing users to exceed their intended level of access.

Retest Results

NCC Group re-tested the 20 findings from the original assessment to verify that they had been adequately fixed. Searching for new, previously undiscovered flaws was not a goal for

the re-test though some time was spent looking for issues similar to the original findings that might have been introduced in version 1.5 or that were missed in the previous assessment. The results of the re-test were:

- 16 of the findings, including all critical- and high-risk issues, were deemed to be fully fixed.
- 3 medium- and low-risk findings were partially fixed. Of those:
  - 1 was fixed with respect to the targets listed in the original findings but NCC Group identified additional targets during re-testing.
  - 2 were partly fixed in ways that greatly reduced the risk posed; while NCC Group recommends additional changes, the risk ratings of the findings were reduced to align with the remaining risk.
- 1 low-risk finding was not fixed. Object First does not deem it to be a priority given their threat model but is considering a fix in a future release.

Strategic Recommendations

- **Advise users to rotate proxy credentials.** Versions of Ootbi prior to 1.5.50.8862 logged HTTP proxy passwords in plain text; those logs may have been shared with Object First through "support bundles". Object First should advise its customers to rotate any proxy credentials that may have been used with the application.
- **Implement privilege separation.** Re-architect the application core so that most of it runs as a low-privilege user with key operations that require additional privileges, such as the following, performed by a high-privilege process:
  - Set the immutability flag and a retention expiry time on a file.
  - Clear the immutability flag on a file only if its retention expiry time has passed.
  - Update the retention expiry time on a file only if the new expiry time is later than the old one.

  Such changes would strengthen Ootbi's immutability properties in the face of exploits that permit arbitrary code execution within the context of the server process.
- **Improve System Hardening for IPMI**. While the physical hardening and security of the appliance against local attackers was not in scope for this engagement and was not explicitly tested, there are several additional protection measures that would improve the platform security against remote attackers when IPMI is enabled. For example, consider setting a BIOS password, enabling Secure Boot, and preventing unauthorized users from modifying kernel boot arguments to boot into single-user mode to ensure the integrity of the platform and prevent unauthorized modifications to the system.
- **Clarify Documentation of Differences Between Ootbi and AWS.** Ootbi implements a subset of the AWS APIs for S3 and IAM and a subset of the AWS permission policy language. Ootbi currently does not support many features and behaviors that users familiar with AWS may expect but not all of these exceptions are clearly documented.
- **Avoid Using Shells to Invoke External Programs.** Shells have complicated syntax and passing potentially unsafe data to them exposes the application to unnecessary risk. Review all code that may invoke external programs, investigate whether user input can reach that code, and where feasible, re-factor to avoid using shells.
- **Sanitize all User Input.** For all API endpoints ensure that user input is correctly sanitized, whether by passing through some sort of encoding such as Base64 or carefully passing user input into the `Process.Start()` method. Removing or deactivating unused endpoints would also make checking user input sanitization much easier.

- **Review the Implementation of AWS Permission Policies.** AWS API authorization can be complicated, and while the application deliberately does not implement many of the more complicated parts, it has a number of flaws and gaps in documentation that may cause application users to write permission policies that grant more access than intended.

2 Dashboard

Target Data

Name	Ootbi Appliance Server
Type	Product Assessment
Platforms	Linux, C#, Python, C++
Environment	Testing

Engagement Data

Type	Product Review
Method	Code Assisted
Dates	2023-12-04 to 2024-08-01
Consultants	5
Level of Effort	54 Person-days

Targets

Management API	Web services API used by the web application UI to communicate with and control the Ootbi cluster.
Ootbi Appliance	Physical server appliance hosting the Ootbi cluster and one or more nodes.
Storage API	Web services API to create buckets and store objects, issue and manage storage API credentials, and manage storage API permissions.
Web UI	Primary web application UI for managing an Ootbi cluster once it is deployed.

Finding Breakdown

	Original Assessment	Remaining after July retest
Critical issues	1	0
High issues	2	0
Medium issues	8	0
Low issues	9	3
Informational issues	0	1

Category Breakdown

	Original Assessment		Remaining after July retest	
Access Controls	8	(H)(M)(M)(M)(L)(L)(L)(L)	1	(L)
Authentication	7	(H)(M)(M)(L)(L)(L)(L)	2	(L)(I)
Cryptography	1	(M)	1	(L)
Data Exposure	2	(M)(L)	0	
Data Validation	2	(C)(M)	0	

Component Breakdown

	Original Assessment		Remaining after July retest	
Management API	5	(C)(M)(M)(L)(L)	2	(L)(L)

Component Breakdown

Ootbi Appliance	6	(H)(M)(M)(L)(L)(L)	6	(L)(I)
Storage API	6	(H)(M)(M)(M)(L)(L)	0	
Web UI	3	(M)(L)(L)	0	

(C) Critical (H) High (M) Medium (L) Low (I) Informational

3 Storage Immutability

The key feature of the Object First Ootbi system is its ability to provide immutable storage. Objects may be assigned retention dates; until the retention date is reached, the file may not be deleted or modified.

Ootbi is designed to be used by Veeam Backup & Replication (VBR) with end-to-end encryption enabled. However, VBR was not in scope for NCC Group's assessment and its behavior was not taken into consideration. NCC Group interacted directly with Ootbi's API without going through VBR.

Threat Model

The Ootbi application is designed to protect against any data breach or malware infestation of an Object First customer: even if all of the customer's secrets, including Ootbi administrator and storage API credentials, are known to the attacker, the attacker still cannot modify data stored within an Ootbi appliance. However, the attacker is assumed to *not* have physical access; an attacker with physical access could obtain a root shell using various techniques or simply destroy an appliance's storage media so Object First is much less concerned with attacks requiring physical access.

Application Design

Application-Level Security

The Ootbi system exposes a subset of the AWS S3 and IAM APIs in order to:

- Create and manage principals with credentials allowing them to interact with the application.
- Assign permissions to principals.
- Create and configure storage "buckets".
- Store, retrieve, and manage objects in those buckets.

This allows customers to interact with it using AWS CLI and other tools and libraries designed to interact with AWS. The chosen subset corresponds to the requirements of Veeam Backup & Replication (VBR) v12; many elements of the AWS API are unimplemented because VBR does not require them.

The Ootbi software can create buckets either with or without "object lock" enabled. Object lock must be explicitly enabled when a bucket is created; there is no way to enable it after bucket creation. There is similarly no way to disable object lock on a bucket. Using AWS CLI, a bucket with object lock can be created using `aws s3api --endpoint-url <address> --bucket <name> --object-lock-enabled-for-bucket`; at the HTTP layer, object lock can be enabled by providing the HTTP header `x-amz-bucket-object-lock-enabled: True` in a request to `PUT /<name>`. Except where otherwise specified, the rest of this discussion only applies to buckets with object lock enabled.

Objects in buckets with object lock are versioned so the application's concepts of deletion and modification of objects are based on object versions. There is no way to modify an existing object, only to create a new version or delete an existing version. Deleting an object (as opposed to deleting a specific version of an object) only creates a new version containing a deletion marker; past versions are retained until explicitly deleted.

Objects in buckets without object lock are not versioned and can be modified or deleted at any time by any principal with the proper permissions. The immutability feature is only relevant to buckets with object lock enabled.

Enabling object lock does not automatically lock object versions stored in a bucket; locking must be explicitly requested[1]. In order to apply a lock to an object version, one must either use the `object-lock-mode` and `object-lock-retain-until-date` options to `s3:PutObject` when creating the object version or call `s3:PutObjectRetention` on an existing object version. Either method allows callers to set a time when retention expires; until that time is reached, the object version is locked and cannot be deleted. It is an error to attempt to set a retention expiry time in the past or that is earlier than the target's existing retention expiry time. It is likewise an error to attempt to set a lock on an object in a bucket that does not have object lock enabled.

Locks and retention times apply only to specific object versions. Locking does not prevent new versions (including deletion markers) from being created; the only operations prohibited against locked versions are those that would delete them or move their retention dates earlier. It is valid for not all versions of an object to be locked or for different versions to have different retention times; in this case, unlocked versions may be deleted while locked ones may not. Notably, if object lock is applied to the most recent version of a deleted object (or is explicitly applied to the version ID of a deletion marker), then the lock only applies to the deletion marker: data versions of that object may still be deleted unless they are also locked.

Operating System-Level Security

The Ootbi appliance uses commodity hardware and a commodity operating system (Ubuntu Linux): it does not provide immutability properties at the physical layer. The lowest level at which immutability is implemented is the filesystem: the underlying files in the XFS filesystem representing object versions get the "immutable" flag (`ioctl(fd, FS_IOC_SETFLAGS, FS_IMMUTABLE_FL)`, or equivalently, `chattr +i filename`) when locked. As per Linux documentation[2], the result of setting this flag is that:

> The file is immutable: no changes are permitted to the file contents or metadata (permissions, timestamps, ownership, link count, and so on). (This restriction applies even to the superuser.) Only a privileged process (CAP_LINUX_IMMUTABLE) can set or clear this attribute.

As a result, a locked object version can only be modified or deleted after a superuser or process with `CAP_LINUX_IMMUTABLE` removes the immutability flag.

The Ootbi appliance does not normally grant shell access to its users (at any privilege level) or give them any other way to directly modify immutability flags. Administrator permissions within the application do not extend to administrator permissions over the operating system. Since the application does need to be able to change immutability of files, both when they are created and when their retention dates are reached, substantial portions of the application *do* run with superuser permissions.

On every attempt to write to or delete an immutable file, its retention expiry time is checked. If that time has passed, the immutability attribute is cleared and the operation is permitted.

1. The application *does* support the API action `s3:PutObjectLockConfiguration` but it has no effect in versions 1.4 or 1.5. Ootbi ignores the optional "Rule" element intended to set default retention periods for all objects stored in the bucket and does not allow object lock to be enabled on an existing bucket.
2. https://man7.org/linux/man-pages/man2/ioctl_iflags.2.html

Otherwise, an error is raised. As a result, filesystem-level immutability is only a safeguard against bugs in the application; in normal operation, the application should never attempt to delete or modify an object whose retention period has not expired. Retention times are stored in extended attributes of locked files. Since the immutability flag also applies to extended attributes, the application needs to remove and replace it whenever it needs to update the retention time of a locked object version.

Customers *can* obtain root shells on the appliance console or via SSH. Normally, obtaining such a shell is only possible using a temporary authorization code issued by Object First as described in the "Support access" section below. It is also possible for a user or attacker with physical access to an appliance to obtain root credentials in other ways, such as by booting an appliance to single-user mode and creating new user credentials that are permitted to elevate to `root`. With access to a root shell, an user or attacker could revoke immutability on files, allowing those files to be deleted or modified, install software back doors, or otherwise compromise the security and immutability guarantees of an Ootbi appliance. The application is not designed to be secure against attackers with physical access to the appliance — an attacker with physical access could simply destroy the storage media, so Object First does not deem defending against attacks requiring physical access to be a priority.

Support Access

Although the appliance console can be accessed via SSH or physical terminal, customers are not given login credentials at any privilege level. In order to log into the appliance console, a customer must use the TTY6 console (either via a direct connection to the physical appliance or as an authenticated user via IPMI, which must be explicitly enabled by an administrator and requires physical access to the machine to obtain the necessary credentials) to request an authorization code from Object First support. Authorization codes are 7-digit numeric codes that are valid for 24 hours and cannot be re-used once entered. Upon entering a correct authorization code, the user is granted access to a root shell on the appliance. Object First reported that generating an authorization code requires manual intervention by multiple members of their support staff.

IPMI is disabled by default. Since enabling IPMI access to TTY6 would make it possible to request an authorization code without physical access to an appliance, customers are advised to not do so. Enabling IPMI requires a password from a physical tag on the appliance.

SSH is disabled by default but can be enabled using either the web or text UI. Since Ootbi appliances are deployed in customer premises, they normally will not be accessible directly from the Internet, though network topology is up to the customer. There is no mechanism for Object First to get a shell or run commands on a deployed appliance, either with or without customer involvement. If a customer needs support, the customer must log into the device using an authorization code (as described above) then perform actions as directed via an out-of-band communication channel (phone, email, etc.).

Due to this design, total compromise of a customer would not allow an attacker to get a shell or install software on an Ootbi appliance as the customer does not have such credentials (so long as the customer has not violated the appliance's protections by enabling IPMI or enabling additional ways to log in). Limited protection is provided against compromise of Object First itself: an attacker could gain the ability to generate authorization codes[3] but would need to either use social engineering techniques to persuade a customer to log into an appliance and perform operations that would compromise its security or

compromise the customer as well in order to enable SSH, log in using an authorization code, and take control of the appliance.

Testing

NCC Group assessed the application's adherence to this design and threat model through a combination of design review, source code review, dynamic application testing, and build review of an appliance using root credentials. Testing activities included the following:

- Reviewing the application's design to understand how immutability is implemented and what operations are intended to be allowed against locked and unlocked objects.
- Reviewing application source code to verify that immutability appeared to be implemented as designed.
- Attempting API requests against locked objects that would violate immutability guarantees if successful.
- Reviewing source code for flaws or undocumented functionality that could permit users, malicious or otherwise, to bypass immutability guarantees.
- Making malformed requests and requests containing unexpected values (including empty inputs, very large inputs, inputs using unexpected and invalid string encodings, invalid dates, inputs containing shell and database query metacharacters, etc.) with the goal of identifying application implementation flaws that could be used to execute arbitrary code within the context of the application.
- Reviewing SSH service configuration to ensure hardening best-practices are being followed, such as disabling root login and TCP forwarding.

Since the application is designed to not permit immutability to be violated by any of its users, authentication and authorization scenarios were not relevant to immutability testing.

The following flaws were identified that could impact data immutability, all of which were fixed as of version 1.5.50.8862:

- In at least one route of the management API (used by the management web UI), unfiltered user input was passed directly into a shell command that ran as root. This flaw could permit arbitrary code execution within an appliance; an attacker with appropriate credentials to the management application could use it to revoke immutability on files (by running "`chattr -i ...`") then delete or modify those files.
- Many other locations in the application's source code unnecessarily used shells to invoke external commands. NCC Group did not locate any other locations where unfiltered user input was incorporated into a shell command but was not able to prove that it was not possible in all locations. NCC Group considers using shells to invoke external commands to be an unsafe programming practice that exposes applications to unnecessary risk and advised that shells not be used to execute external commands.

The following possible risks were not tested:

- The behavior of the underlying Ubuntu Linux operating system, which NCC Group assumed to function as documented for its configuration.
- Low-level TOCTTOU attacks involving changes to file extended attributes and immutability flags.

3. Object First reports that no one person can generate and release an authorization code; at least two different authorized support personnel must be involved. Consequently, an attacker may need to compromise Object First at multiple points in order to generate its own authorization codes.

- Attacks based on desynchronizing appliance time. Since the application's immutability behavior depends on an accurate system clock, it may be possible to tamper with NTP or NTS servers or responses to advance the appliance's clock past the retention time for object versions that an attacker wishes to delete.
- The security of the Object First-controlled `apt` repository used by Ootbi appliances to obtain updates to system and Object First software.
- The integrity of the build processes used by Object First to create appliance images and software updates.
- Object First's processes for issuing root shell authorization codes and for verifying customer identities before doing so.

NCC Group's testing was not limited to verifying the immutability properties of Object First Ootbi. Testing efforts included many other aspects of the application and identified a number of other flaws that did not affect immutability.

4 Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors.

AWS API

Title	Status	ID	Original Risk	Remaining Risk
Application Disregards Service Names During Authorization	Fixed	HKX	High	None
Resource Constraints Not Supported For IAM API	Fixed	CWC	Medium	None
Resource Constraints Not Supported By `s3:CreateBucket`	Fixed	9XB	Medium	None
Incorrect Permission Checks For `s3:CreateBucket`	Fixed	DRG	Medium	None
Time-Variant Signature Comparison Leaks Information	Fixed	M3J	Low	None
Application Accepts Unsigned Parameters	Fixed	7FN	Low	None

Management API

Title	Status	ID	Original Risk	Remaining Risk
OS Command Injection Leading to Remote Code Execution	Fixed	6XC	Critical	None
External Commands Executed As Shell Commands	Fixed	HT6	Medium	None
Unsafe Password Handling	Partially Fixed	C2A	Medium	Low
System User Password Update Does Not Require Re-Authentication	Partially Fixed	REA	Low	Low
Insecure Direct Object Reference Leading to Arbitrary Account Takeover	Fixed	7JP	Low	None

Object First Server

Title	Status	ID	Original Risk	Remaining Risk
Weak Default Credentials	Partially Fixed	Y2P	High	None
Proxy Password Written to Logs	Partially Fixed	7BK	Medium	None
Proxy Credentials Included In Support Bundle	Fixed	2KR	Medium	None
SSH Server Allows TCP Forwarding	Fixed	TGT	Low	None

Title	Status	ID	Original Risk	Remaining Risk
SSH Server Allows Password Authentication	Partially Fixed	GAE	Low	Info
Local/SSH Authentication Does Not Support MFA	Not Fixed	JER	Low	Low

Web UI

Title	Status	ID	Original Risk	Remaining Risk
Unauthenticated Support Bundle Download	Fixed	ADA	Medium	None
Client-Side Security Controls for IP Whitelist	Fixed	NWJ	Low	None
Insecure HTTP Caching Controls	Fixed	PCN	Low	None

5 Contact Info

The team from NCC Group has the following primary members:

- Andrew Havard – Consultant
  andrew.havard@nccgroup.com
- Rennie deGraaf – Consultant
  rennie.degraaf@nccgroup.com
- Vishtasp Jokhi – Consultant
  vishtasp.jokhi@nccgroup.com
- William Enright – Consultant
  william.enright@nccgroup.com
- William Groesbeck – Consultant
  william.groesbeck@nccgroup.com
- Kivanc Tos – Project Manager
  kivanc.tos@nccgroup.com
- Marina Pinzaru – Account Manager
  marina.pinzaru@nccgroup.com

The team from Object First has the following primary members:

- Alexey Podoselnik
  alexey.podoselnik@objectfirst.com
- Eric Schott
  eric.schott@objectfirst.com