TrustedCI: The NSF Cybersecurity Center of Excellence
Open OnDemand Report

December 31, 2018

For Public Distribution after May 2019

Elisa Heymann[1], Joel Atkins[2], Barton P. Miller[3]

[1] Software Assurance Lead, elisa@cs.wisc.edu
[2] Student Researcher, atkins@cs.wisc.edu
[3] Co-PI bart@cs.wisc.edu

About Trusted CI

Trusted CI is funded by NSF's Office of Advanced Cyberinfrastructure as the NSF Cybersecurity Center of Excellence (CCoE). In this role, Trusted CI provides the NSF community a coherent understanding of cybersecurity's role in producing trustworthy science and the information and know-how required to achieve and maintain an effective cybersecurity program. Trusted CI achieves this mission through a combination of one-on-one engagements with NSF projects, training and best practices disseminated to the community through webinars, and the annual, community-building NSF Cybersecurity Summit for Large Facilities and Cyberinfrastructure.

Acknowledgments

Trusted CI's engagements are inherently collaborative. The authors wish to thank the Open OnDemand team, and specifically Trey Dockendorf[4], for the collaborative effort that made this document possible. The Open OnDemand is supported by NSF awards PHY-1534949[5] and PHY-1835725[6].

This document is a product of Trusted CI. Trusted CI is supported by the National Science Foundation under Grant Number ACI-1547272[7]. For more information about Trusted CI please visit https://trustedci.org . Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Using & Citing this Work

Cite this work using the following information:

Elisa Heymann, Joel Atkins, Barton P. Miller. "TrustedCI: The NSF Cybersecurity Center of Excellence Open OnDemand Report". TrustedCI: The NSF Cybersecurity Center of Excellence. December 2018.

[4] OSC HPC Systems Engineer, tdockendorf@osc.edu
[5] https://www.nsf.gov/awardsearch/showAward?AWD_ID=1534949
[6] https://www.nsf.gov/awardsearch/showAward?AWD_ID=1835725
[7] https://www.nsf.gov/awardsearch/showAward?AWD_ID=1547272

Table of Contents

List of Figures

Executive Summary

The Open Supercomputing Center (OSC) and Trusted CI collaborated to assess the security of Open OnDemand (OOD). Open OnDemand is an HPC portal based on OSC's original OnDemand portal. The goal of Open OnDemand is to provide an easy way for system administrators to provide web access to their HPC resources, including, but not limited to:

- Plugin-free web experience.
- Easy file management.
- Command-line shell access.
- Job management and monitoring across different batch servers and resource managers.
- Graphical desktop environments and desktop applications.

As wider communities are evaluating the Open OnDemand software and collaborating with OSC, a security assessment becomes an important aspect of the software.

This assessment was based on a First Principles Vulnerability Assessment (FPVA)[8] analysis of Open OnDemand. This assessment started by mapping out the architecture and resources of the system, paying attention to trust and privilege used across the system, and identifying the high value assets in the system. From here, a detailed study of the high-risk portions of the code was made. As areas of high risk were studied in detail, we have increased confidence in the security of the code. Results at each step of the process were shared with the Open OnDemand development team.

Major findings include:

- Discovered several implementation issues (bugs) that could affect the proper operation of Open OnDemand.
- No major issues were found in a comprehensive evaluation of the architecture and critical resources in Open OnDemand, and analysis of potential code weaknesses in critical components.
- Found a potential vulnerability, dormant based on current configuration and launch settings.

[8] James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann, "First Principles Vulnerability Assessment", *2010 ACM Cloud Computing Security Workshop (CCSW),* Chicago, IL, October 2010.

1 Overview

1.1 Introduction

This document describes the Trusted CI - Open OnDemand engagement, which occurred July 2018 to December 2018. The goals of the engagement were to evaluate the technology and architecture of the Open OnDemand middleware, perform a code-level security review of the Open OnDemand software, and provide comprehensive reports for any vulnerabilities identified.

This document is organized by tasks as they occurred chronologically, beginning with determining the scope of the engagement and ending with appendices of the results of the various test/analyses of the code and the resulting recommendations for remediation of issues.

1.2 Background

Open OnDemand[9] is open source software, developed by the Ohio Supercomputing Center, which provides a web-based graphical interface access to HPC (High Performance Computing) resources. The software allows the user to login to using their HPC credentials and access resources entirely within their web browser. The core installation includes applications for Shell Access, Job Editing and Creation, File Management and Job Tracking. In total, the components of Open OnDemand are composed of approximately 50,000 lines of code.

1.3 Methodology

This engagement focused on performing a First Principles Vulnerability Assessment (FPVA)[10] on Open OnDemand. FPVA is a methodology for vulnerability assessments developed at the University of Wisconsin-Madison.

Dave Hudak[11] applied for a Trusted CI engagement for the Open OnDemand project in early 2018. He stated that the application was intended to strengthen software assurance for current users, as well as to increase Open OnDemand's "adoption rate in the security conscious HPC community."

1.4 Logistical Issues

[9] http://openondemand.org/
[10] http://research.cs.wisc.edu/mist/VA.pdf
[11] OSC Executive Director, dhudak@osc.edu

The in-depth software engagement differs in several ways from most previous engagements performed by TrustedCI. Open OnDemand was only the second in-depth software engagement done by TrustedCI. There were a few initial issues that slowed this task, primary these were delays in establishing a fully-functional software testbed.

The engagement began by working with the Open OnDemand team to establish Virtual Machines which simulated a typical installation of the software. Several issues arose during this process, such as VM software version compatibilities and VM file system configurations. Iterative updates to the VM configurations and installation process lead to a functional testbed, only to run into issues with the system not functioning after a restart. This was followed by another round of collaboration with the OOD Development team resulting in a final update to the VM configuration and a reliable testbed installation on 1 August 2018.

In early September 2018, conversations between the assessment team and the OOD team focused on the shortcomings of the VM testbed, specifically features of a typical installation of the Open OnDemand software that were not present in the testbed. In response, the assessment team was given credentials for two OSC-maintained instances of the OOD software. The first being their production deployment and the second an internal testbed. In addition, work began on an additional testbed on which the assessment team would have an account with elevated privileges in order to more thoroughly test the software. This additional testbed was made available to the assessment team on 28 November.

2 The First Principles Vulnerability Assessment Methodology

First Principles Vulnerability Assessment (FPVA) is an analyst-centric (manual) methodology that aims to focus the analyst's attention on the parts of the software system and its resources that are most likely to contain vulnerabilities that would provide access to high-value assets. FPVA finds new threats to a system and is not dependent on a list of known threats. The FPVA methodology consists of five steps for evaluating a given piece of software.

1. **Architectural Analysis:** determine the major structural components of the system and then how they interact. At this point architectural diagrams are produced which illustrate the structure of the system.

2. **Resource Identification:** identify key resources accessed by each component. Examples of these resources are files, databases, logs and devices. Resource diagrams are produced that illustrate these resources and their connection to system components.

3. **Trust and Privilege Analysis:** identify the trust assumptions about each component, answering such questions as how are they protected and who can access them? Associated with trust is describing the privilege level at which each executable component runs. The artifact produced at this stage is a further labeling of the basic diagrams with trust levels and labeling of interactions with delegation information.

4. **Component Evaluation:** examine relevant components in depth. A key aspect of the FPVA process is that this step is guided by information obtained in the first three steps, helping to prioritize the work so that high value targets are evaluated first. Any vulnerabilities identified result in the production of a comprehensive vulnerability report that is disseminated to the requesting parties. All work done during this step is logged for inclusion in the final report.

5. **Dissemination of Results:** a final document, this report, is then prepared and includes the deliverables mentioned above as well as an outline of the work completed. Identified vulnerabilities are included as well as areas that have been investigated but no vulnerabilities found. This report is then disseminated to the requesting parties (i.e., the lead of the development team).

These steps were adhered to in the Open OnDemand engagement.

We note that Open OnDemand is a large and complex software system, so no assessment activity will be able to find all possible sources of insecurity. Regular assessments of the software will help maintain its security. In addition, Open OnDemand uses a variety of software technologies resulting in a complex software stack. As such, there needs to be ongoing attention to security of the external software on which Open OnDemand depends.

2.1 Architectural Analysis

Open OnDemand documentation was analyzed followed by the establishment of an Open OnDemand testbed setup upon which to perform the evaluation. Based upon our study of the documentation and testbed we produced an architectural diagrams, to provide a graphic overview of the structure of a standard Open OnDemand installation. Figure 1 shows an overview of the architecture of a running dashboard host, with the more detailed diagrams shown in Appendix A.

From further study of code and documentation, we produced a series of architectural diagrams that provide detailed information regarding component interaction during various use cases of the Open OnDemand code base.

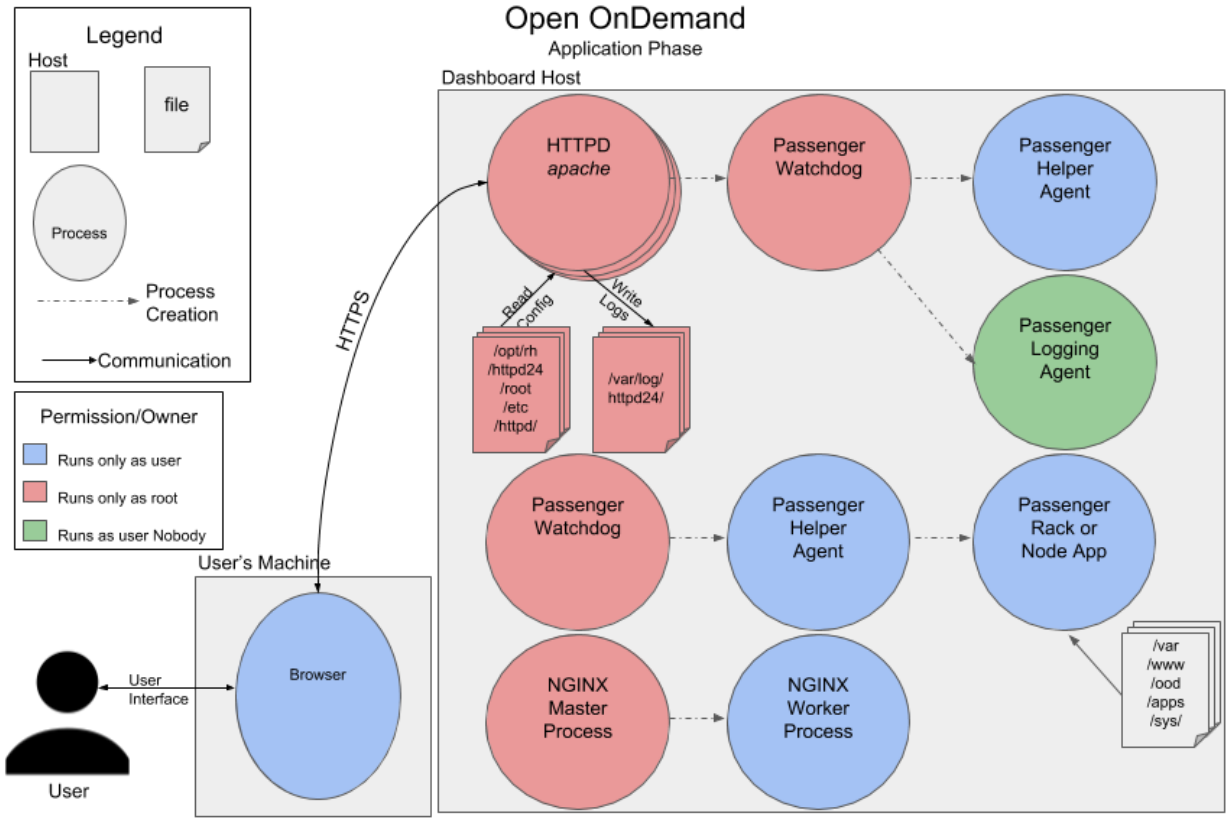


Figure 1. Open OnDemand Startup Phase Process Creation.

From these architecture diagrams we identified the potential attack surface. This surface consists of user inputs to a webpage view. We then identified the path (attack vector) any user input must take to reach any potential impact surfaces. As the assessment was focused upon Open OnDemand as a whole, we were able to identify its individual components and how they interact with one another, as well as how they interact with the user and HPC resources. This

information permits more focused investigation in later steps when evaluating code or designing test inputs. The diagrams also illustrate the separation between various components and show that individual users are isolated from one another.

2.2 Resource Identification

Following the production of the architectural diagrams, the evaluation team identified the key resources accessed by the components identified above. This information was used to produce system Resource Diagrams, such as the HTTPD diagram in Figure 2. The full collection of resource diagrams produced are shown in Appendix B.

From these resource diagrams we are able to identify potential problems for further investigation, such as log files which were evaluated in the Component Analysis step for possible overflows.

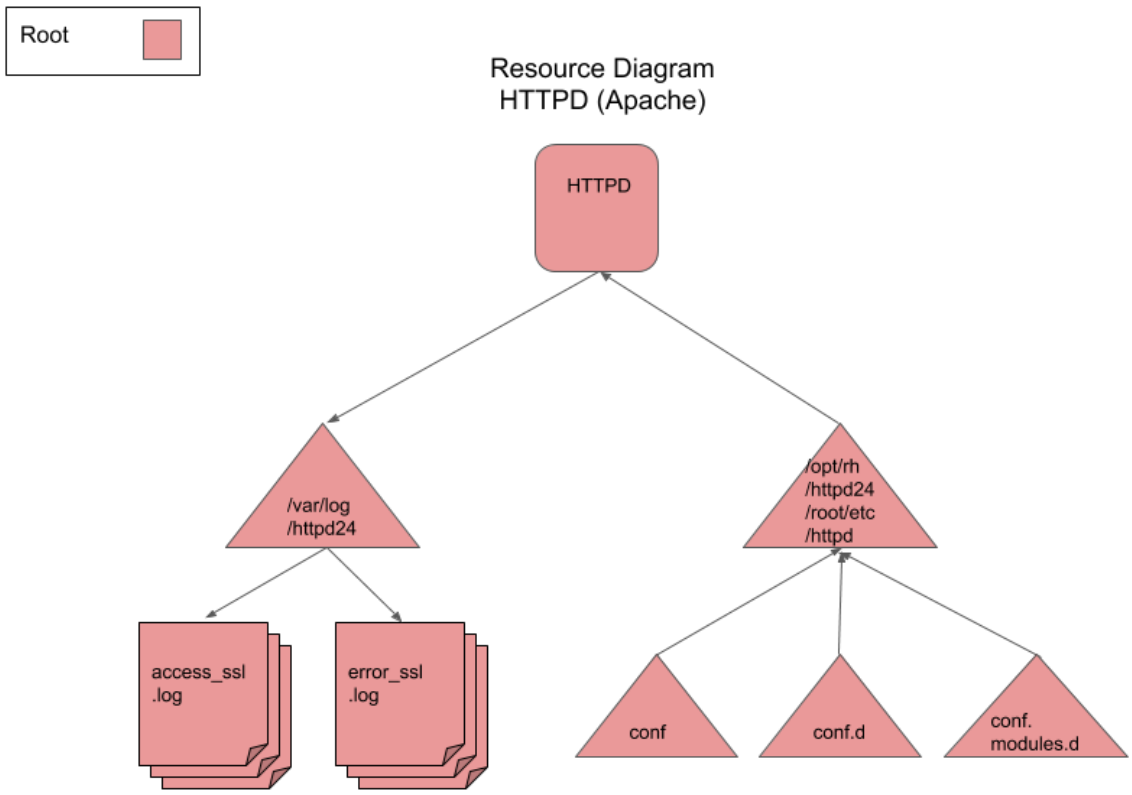


Figure 2. Resource Diagram for Open OnDemand HTTPD Installation.

2.3 Trust and Privilege Analysis

The architecture and resource diagrams, described above, were augmented to describe the various levels of privilege at which Open OnDemand components run. From these diagrams, we can see that Open OnDemand HTTPD and NGINX run with root privileges, but users do not have direct interaction with these processes.

2.4 Component Evaluation

This section describes some of the areas of focus for the component analysis where vulnerabilities are searched based on the previous steps in the analysis. At this step, we evaluated the implementation details looking for weaknesses that might be exploited.

This section describes bugs that need to be addressed, and areas that were evaluated and which appeared to have no issues.

2.4.1 Security of Server Communication

Action: Because of the current configuration this is not of immediate concern. If a future configuration change moves one of the processes out of the same protection environment then this connection will become vulnerable. Recommend encrypting this connection.

Communication between the user and the Open OnDemand dashboard host are encrypted with the HTTPS protocol. Conversations with the OOD team highlighted the fact that communication between the OOD host and login nodes serving interactive applications takes place without encryption. This is a low level concern, as accessing the traffic would require having compromised either end of the communication, but one which the OOD team has identified as a target for future improvement.

Figure 3 shows intercepts of unencrypted traffic from the ood-test dashboard host to a login node running a Jupyter server.

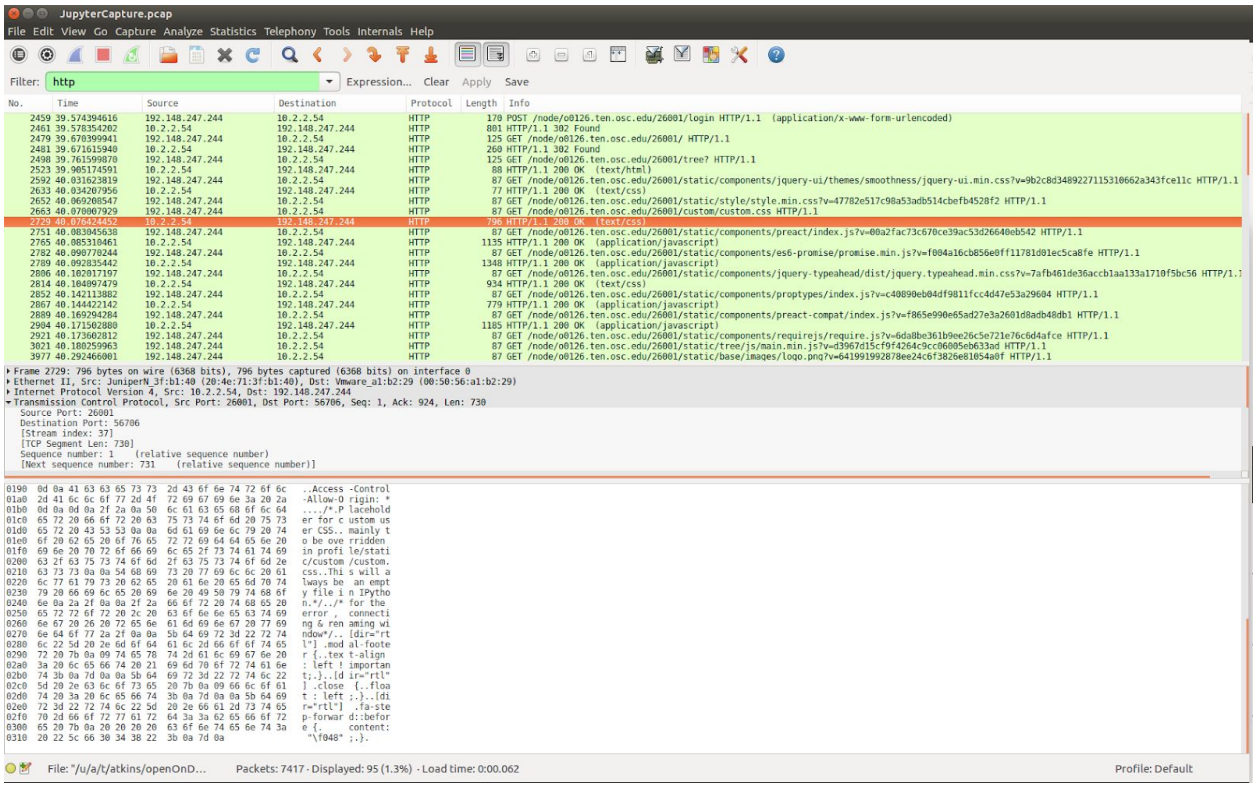


Figure 3. Unencrypted Communications between Servers

2.4.2 Bug - Default Job Template Failure

Action: Bug that affects the correct operation of Open OnDemand and should be fixed.

The default “Simple Sequential Job”, as shown in Figure 4 fails as there are not execution permissions on the job script. The result is shown in Figure 5.

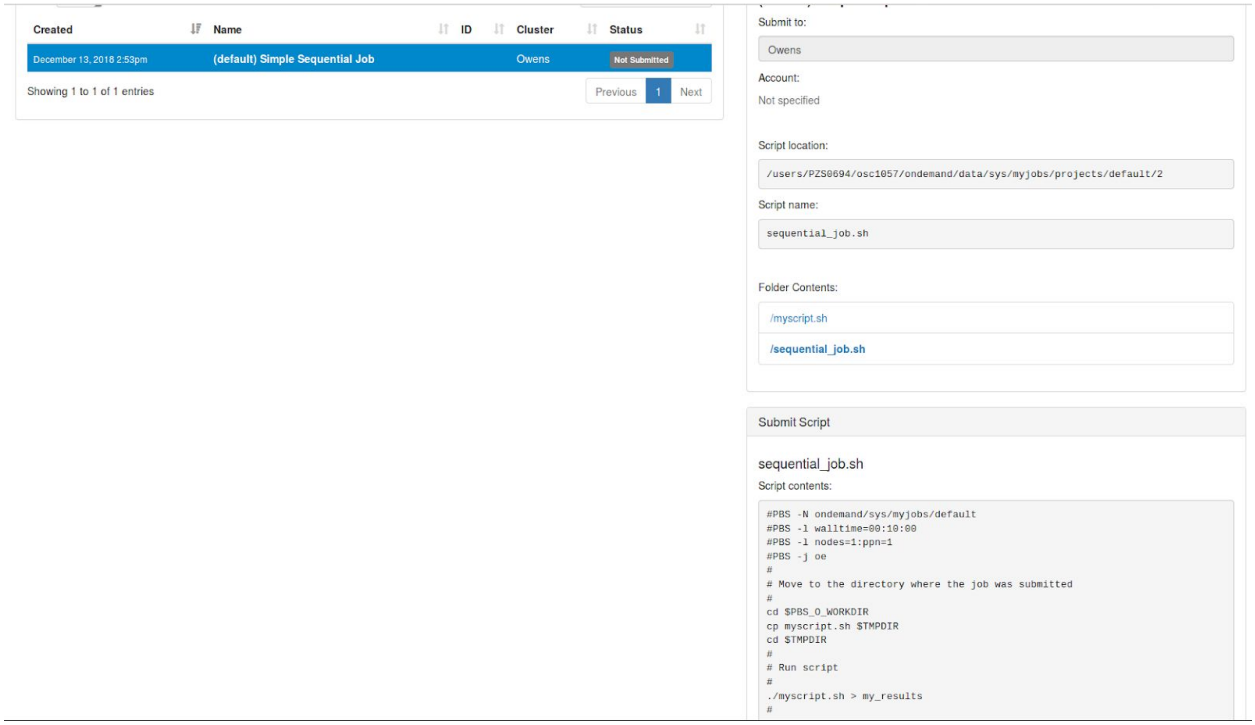


Figure 4. Default Sequential Job Template

Figure 5. Permissions Error

2.4.3 Input Validation

Action: Bugs that affect the correct operation of Open OnDemand and should be mitigated.

We identified multiple issues with unusual inputs causing unintended changes to webpage display.

2.4.3.1 Long Job Name in Composer Moves Elements from View

Log job names move columns out of view in Job Composer. These items are then inaccessible.

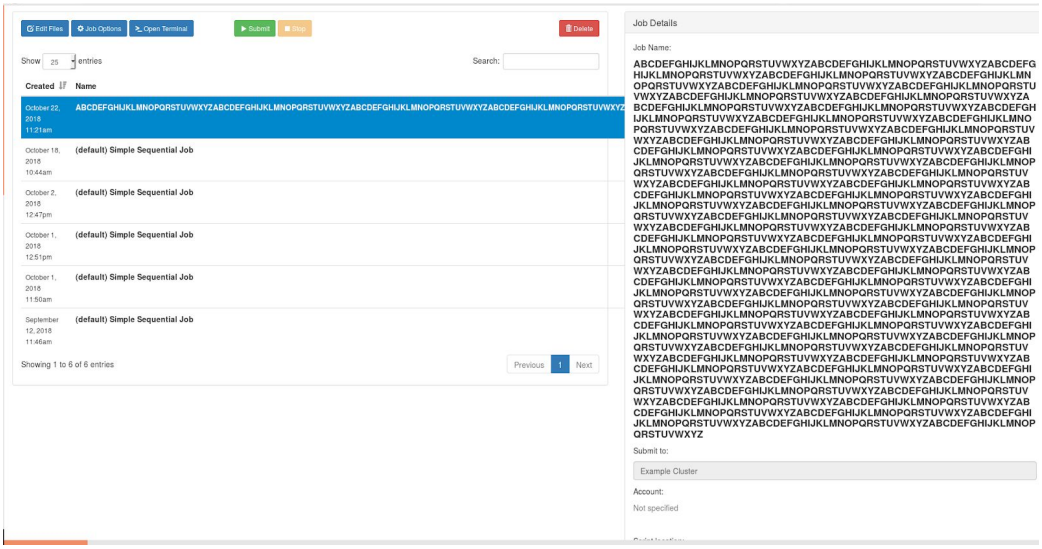


Figure 6. Elements Moved from View

As seen in Figure 6, multiple columns are removed from view. These are displayed in Figure 7 .

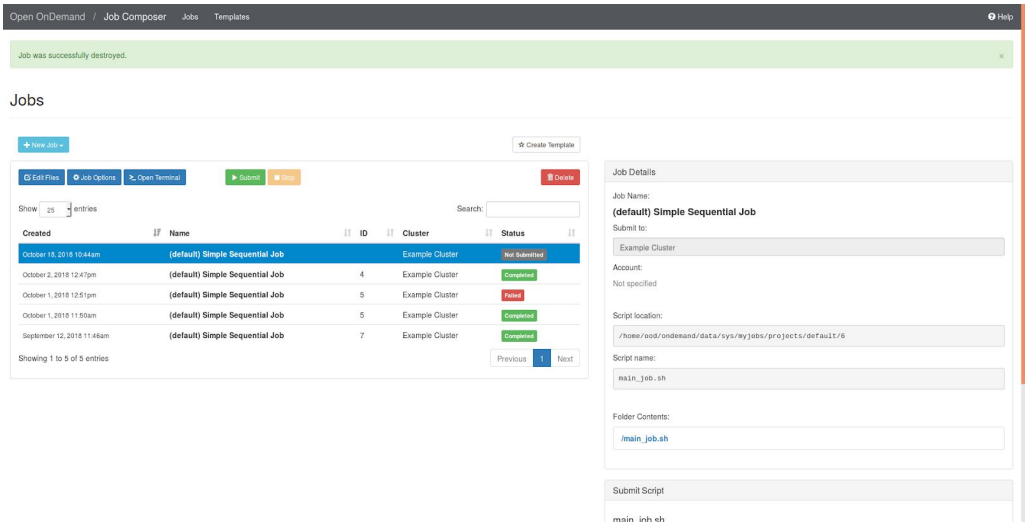


Figure 7. Intended Element View

2.4.3.2 Invalid Characters in New Directory Command of File Explorer

Entering invalid characters into the File Explorer's prompt for creating a new directory results in unintended directory structure changes.

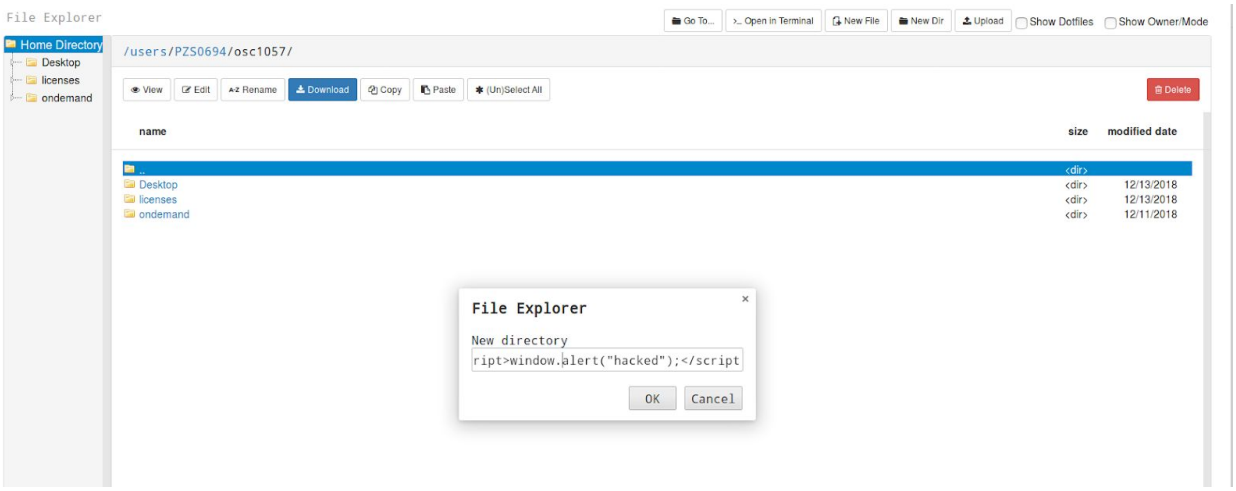


Figure 8. Invalid Directory Name

Entering the code in Figure 8 results in the creation of multiple subdirectories, some of which are inaccessible, as shown in Figure 9.

Figure 9. Inaccessible Directory Structure

2.4.4 Job Limiting

Action: A detailed evaluation was performed and no apparent issues found.

As the resources are shared, the ability of a single user to consume all available resources were tested. This is generally handled by the workload manager. The VM testbed did not account for this and an individual user could continuously spawn processes until all resources were consumed. Conversations with the development team identified this as a shortcoming of the VM testbed and ultimately lead to the use of the testbeds described in Section 1.4, which were located at the Ohio Supercomputing Center. In a production environment Unix Control Groups (Cgroups) are used to prevent this.

2.4.5 User Isolation

Action: A detailed evaluation was performed and no apparent issues found.

The ability of one validated user to affect the operations or data of another user was investigated. The OOD architecture is intended to prevent these sorts of attacks through the use of Per-User NGINX (PUN) instances. The use of PUNs isolates users from one another. The OOD file explorer, as currently deployed, allows users to view directories of other users, which could leak information, but permissions prevent them from modifying the contents of those directories.

2.4.6 Authentication

Action: A detailed evaluation was performed and no apparent issues found.

Open OnDemand various authentication mechanisms. These are third-party mechanisms developed to extend the functionality of the widely used Apache web server software. Authentication done through Apache is also mapped onto HPC resources through the ood_auth_map package developed by OOD. This process was investigated for security and possible information leakage with no vulnerabilities found.

2.4.7 Log Overflows

Action: A detailed evaluation was performed and no apparent issues found.

The possibility of overflowing log files was investigate. Open OnDemand utilizes the logging functions native to Apache and NGINX (with Passenger logging its activity to the NGINX logs). These functions have protections against log overflows and include the creation of compressed log backups when logs reach a certain size, thus preventing log overflows.

2.4.8 XSS and CSRF

Action: A detailed evaluation was performed and no apparent issues found.

As Open OnDemand is a web based interactive portal, the common web vulnerabilities of Cross Site Scripting (XSS) and Cross Site Request Forgeries (CSRF) were evaluated. Open OnDemand properly validates user inputs to prevent XSS attacks and CSRF attacks are prevented by properly implemented user authentication and tokens.

2.4.9 Use of all available resources

Action: Verify that the vulnerability is mitigated in the testbeds where Open OnDemand is deployed, by using cgroups, for example.

When submitting a job, the user can craft a tasks that continuously spawns new processes and consumes all available resources on the executing node. This can result in a Denial of Service (DoS) on the executing node.

The details of this vulnerability are described in the OpenOnDemand-2018-0001 document, attached to this report.

Appendices

Appendix A: Architectural Diagrams

A.1 List of Architecture Diagrams for Open OnDemand

Architectural diagrams were created for a representative Open OnDemand installation. This consisted of an Open OnDemand web portal and HPC login/compute nodes. The diagrams represent the states and communication between services during various phases of job submission and execution.

Diagrams have been created for:

1. Startup of Open OnDemand.
2. Processes on Dashboard Host.
3. Shell Program Processes and HPC Communication.
4. User Isolation through PUNs
5. Unencrypted Traffic Between Dashboard Host and Login Node.

A.2 Architecture Diagrams

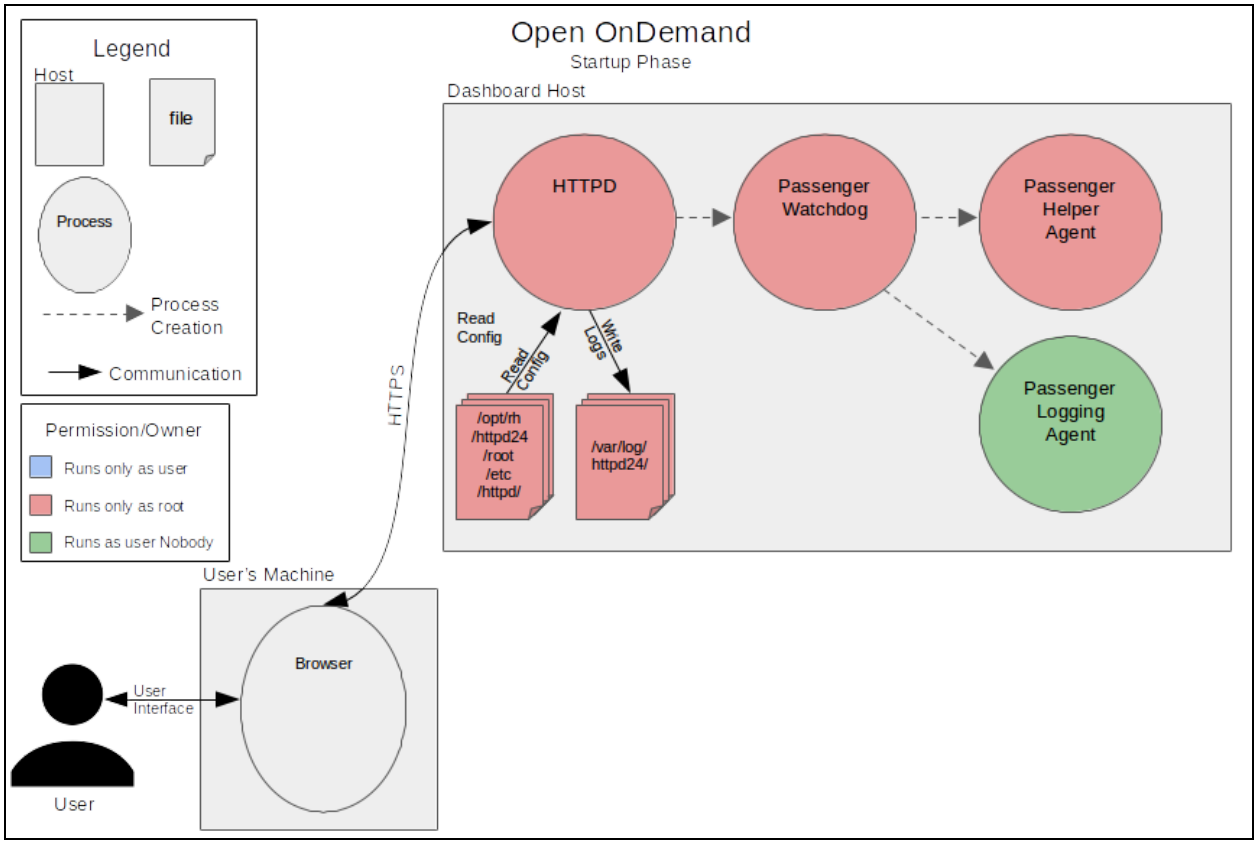


Figure 10. Architectural Diagram, Startup of Open OnDemand.

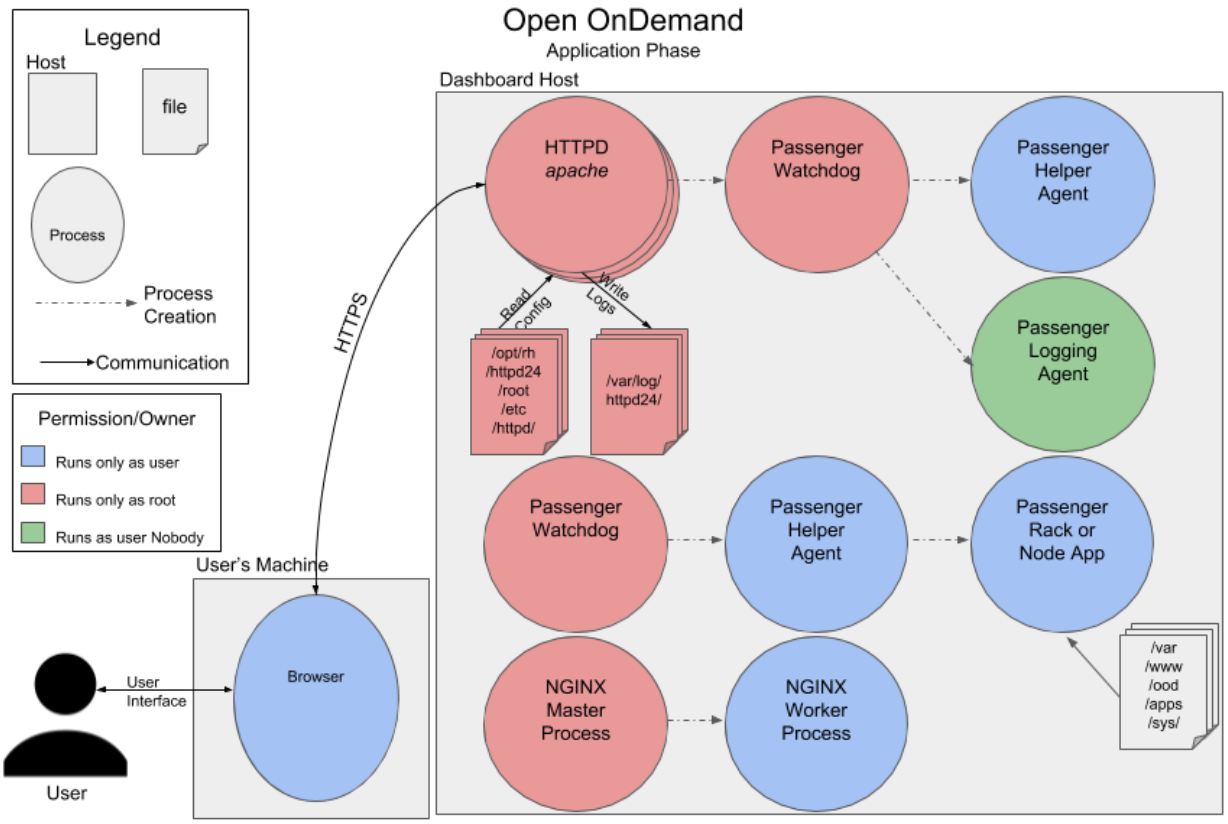


Figure 11. Architectural Diagram, Processes created on Dashboard Host.

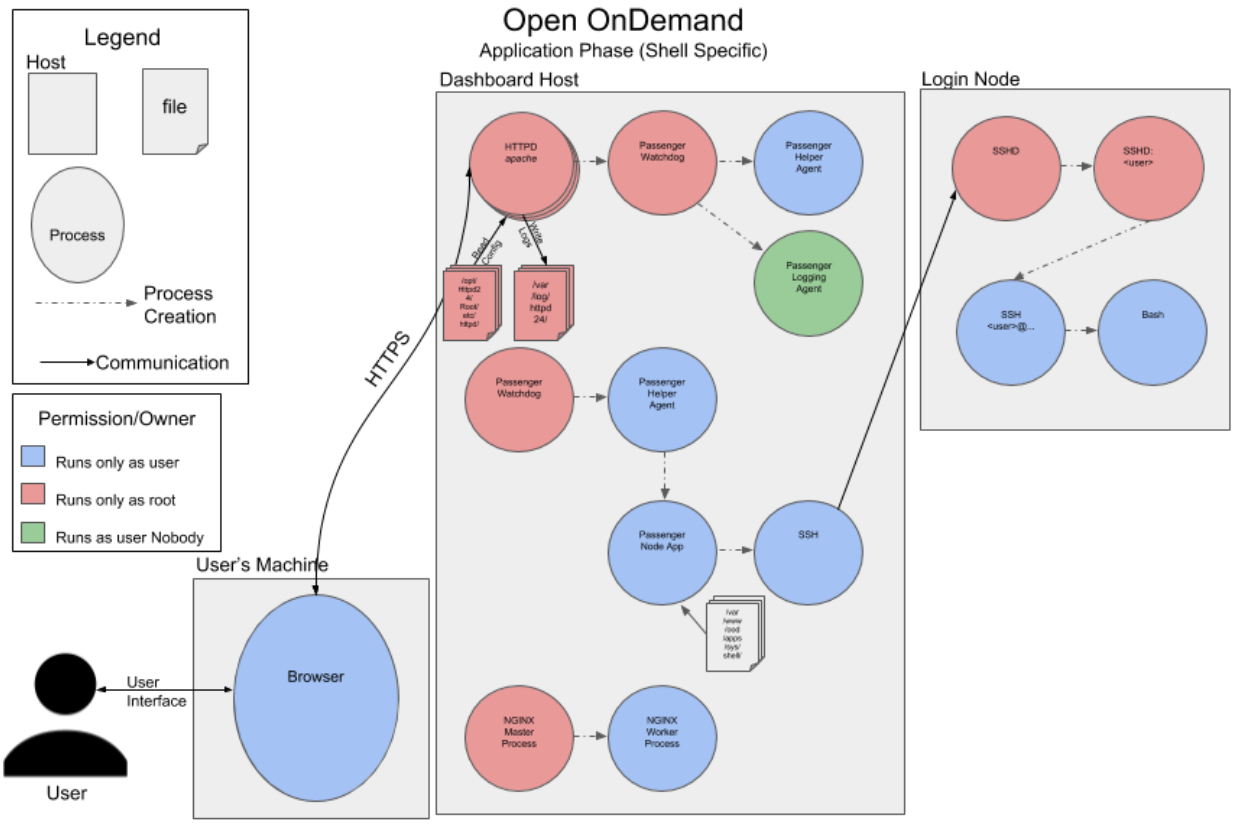


Figure 12. Architectural Diagram for Shell Application.

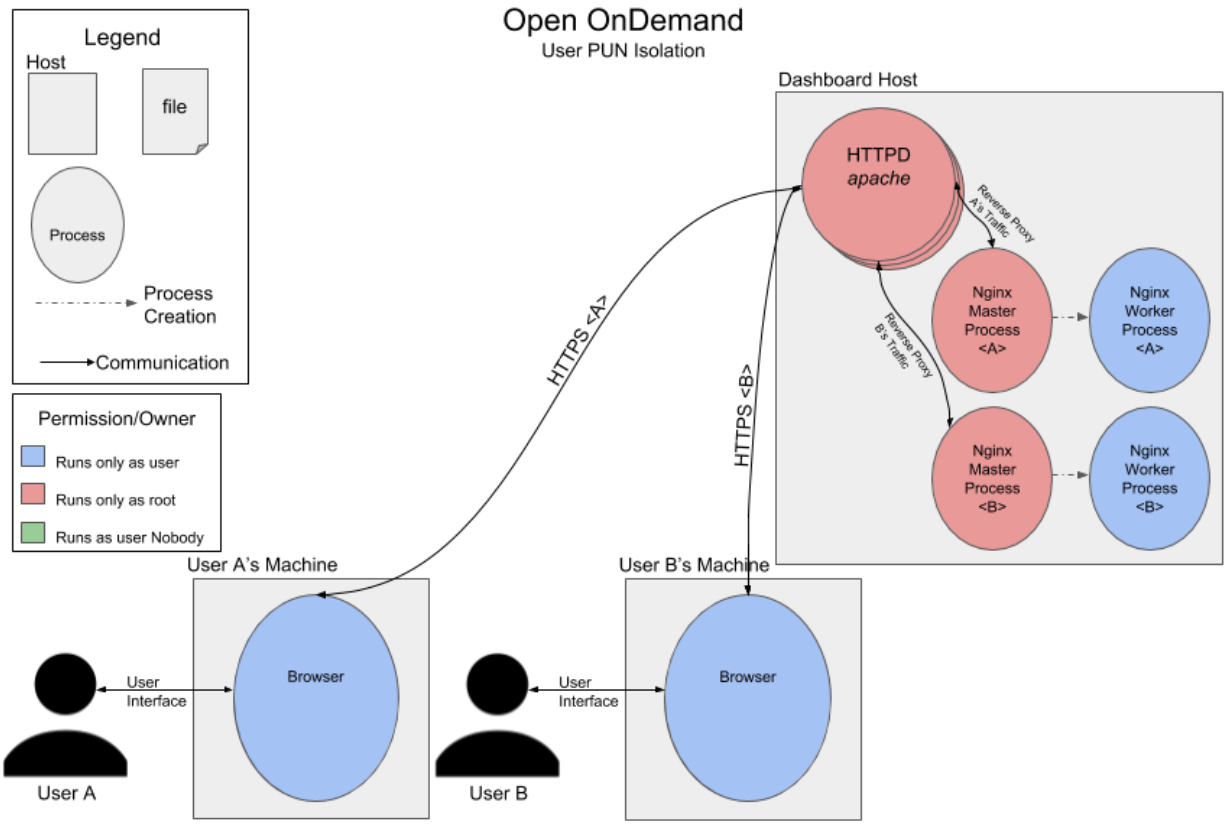


Figure 13. Architectural Diagram for User Isolation through PUNs.

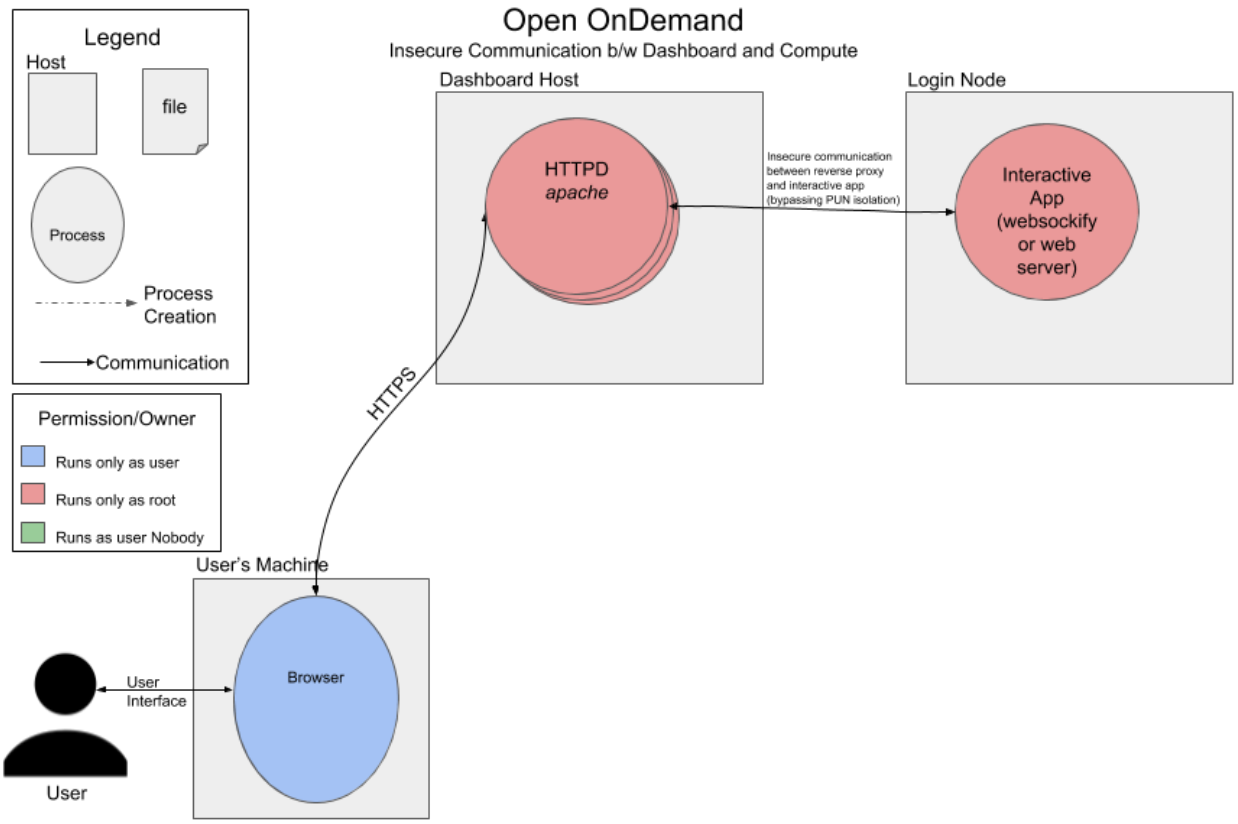


Figure 14. Architectural Diagram demonstrating Unencrypted Traffic between Dashboard and HPC Login Node.

Appendix B: Resource Diagrams

B.1: List of Resource Diagrams for Open OnDemand

Resource diagrams were created for various processes in the Open OnDemand pipeline. 3rd Party web servers (Apache and NGINX) handle communications, Open OnDemand Infrastructure code configures the OOD instance, Open OnDemand Applications are the interactive web pages served to the user.

Diagrams have been created for:

1. Open OnDemand HTTPD Resources.
2. Open OnDemand NGINX Resources.
3. Open OnDemand Phusion Passenger Resources.
4. Open OnDemand Infrastructure and Application Resources.

B.2: Resource Diagrams

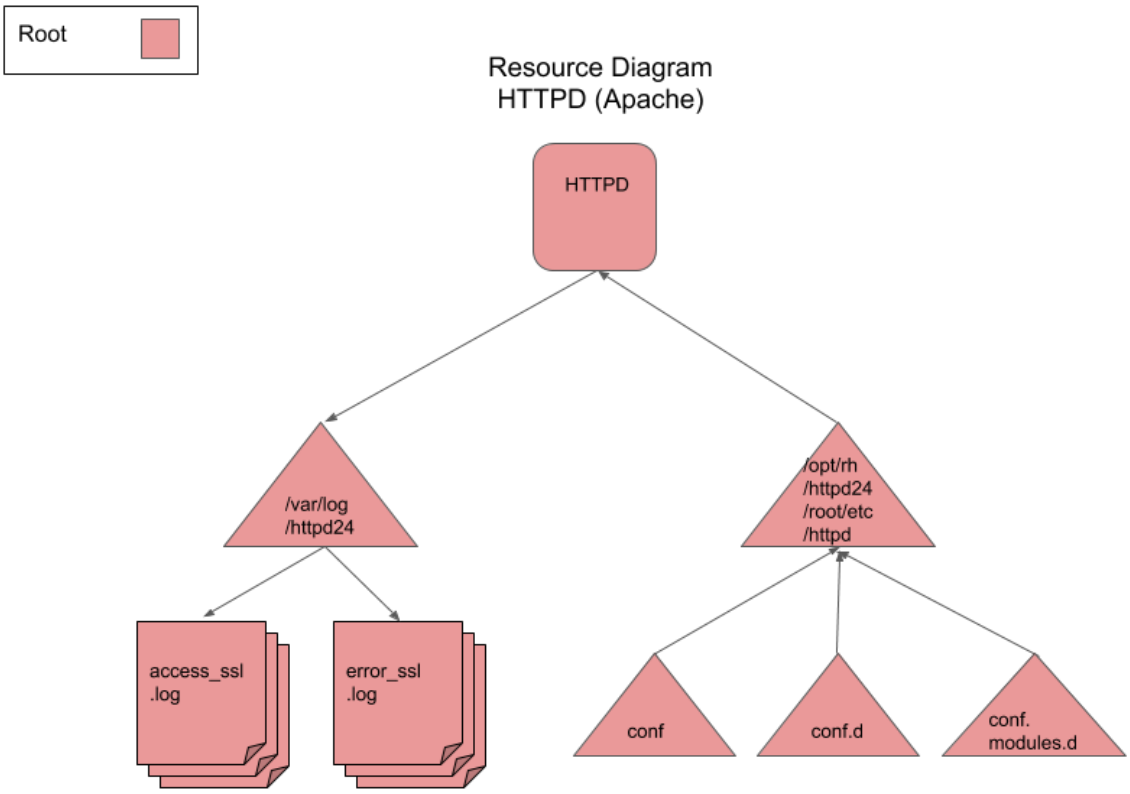


Figure 15. Resource Diagram for Open OnDemand HTTPD Installation.

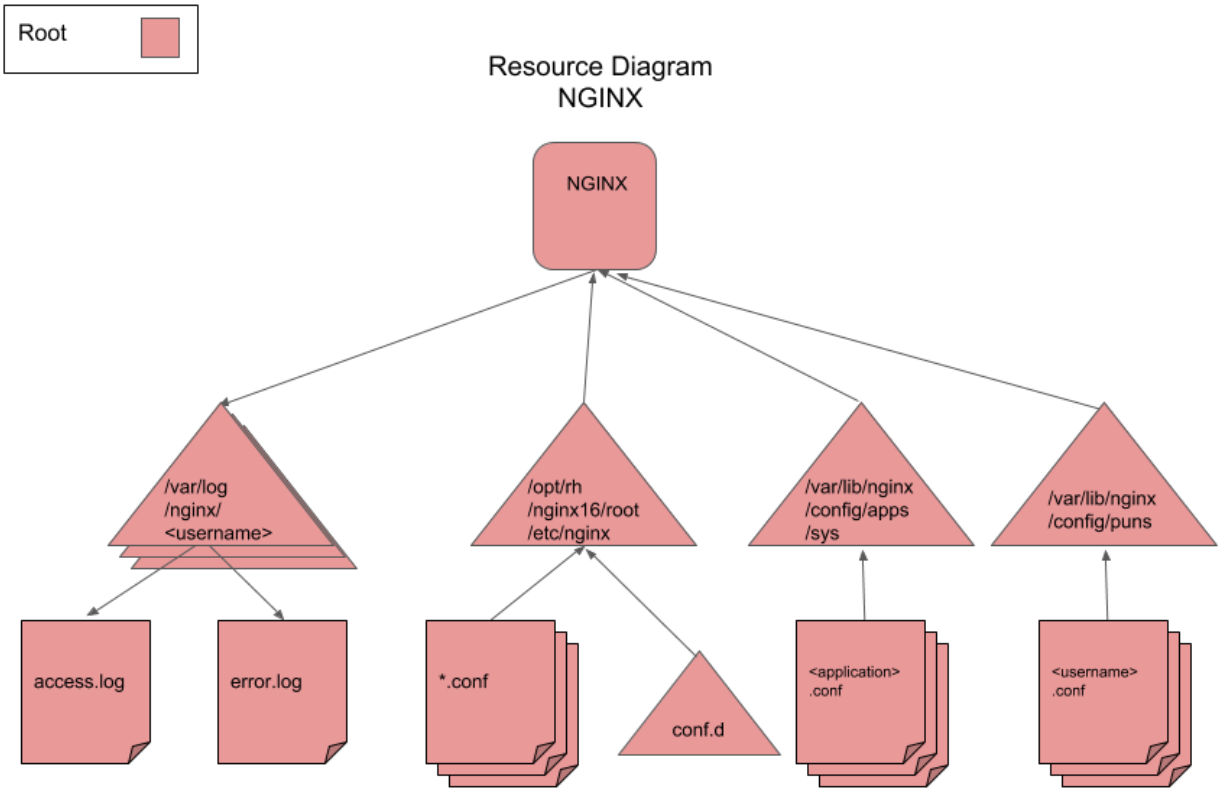


Figure 16. Resource Diagram for Open OnDemand NGINX Installation.

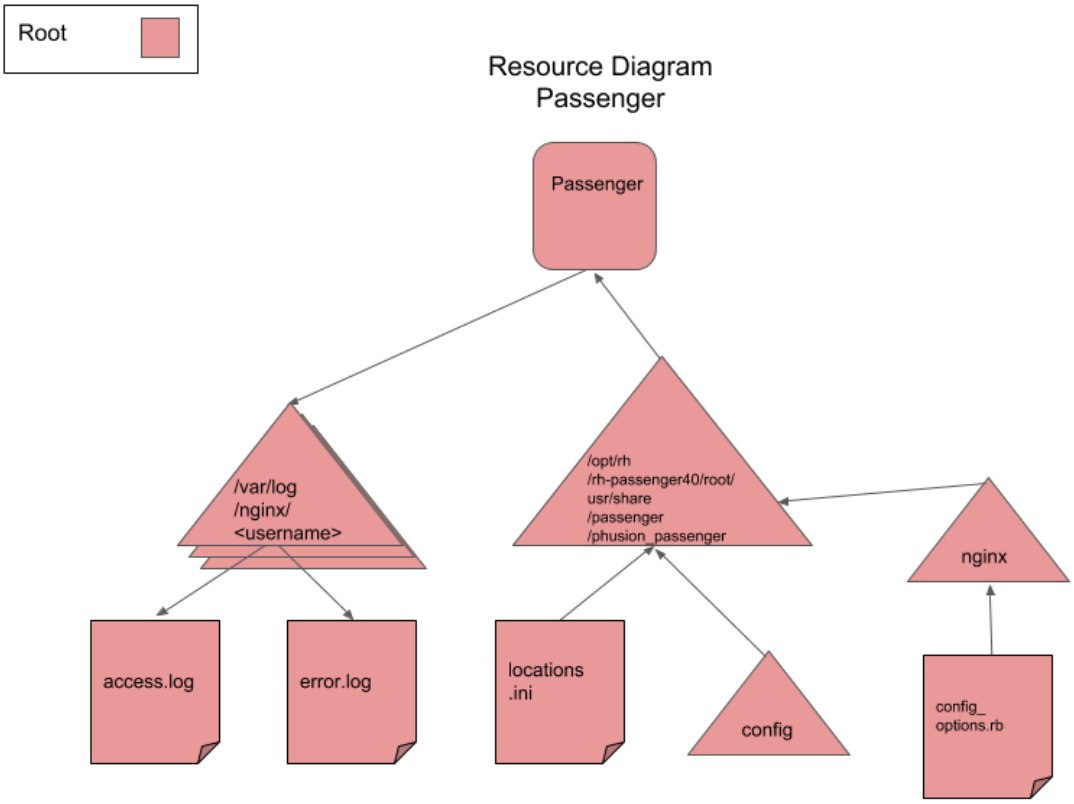


Figure 17. Resource Diagram for Open OnDemand Phusion Passenger Installation.

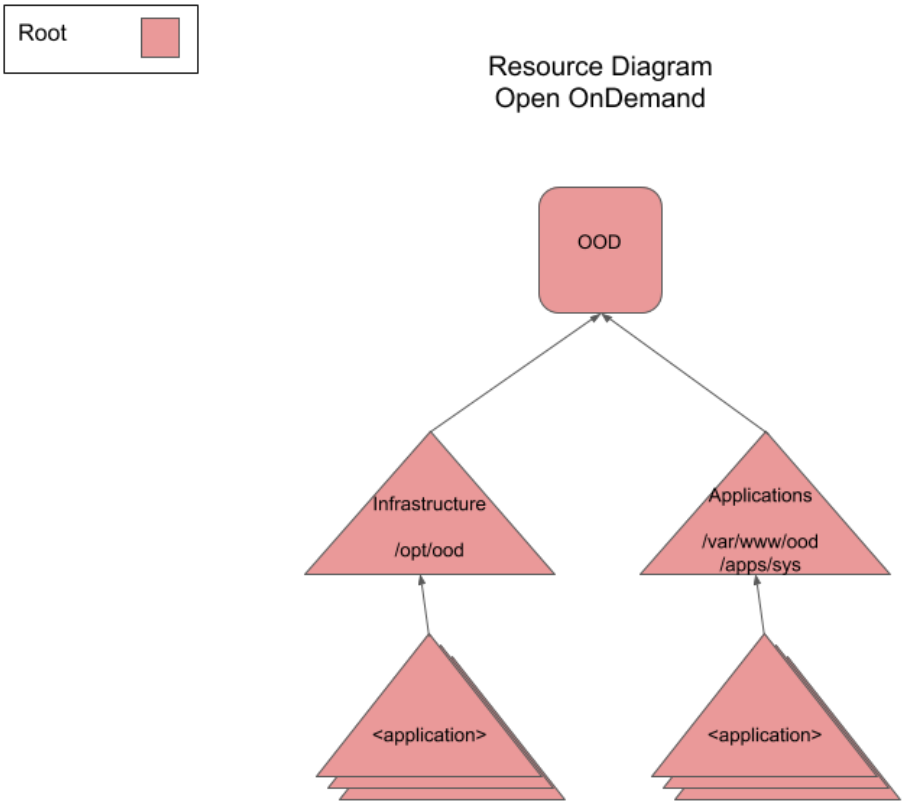


Figure 18. Resource Diagram for Open OnDemand.

Appendix C: Vulnerability report OpenOnDemand-2018-0001

OpenOnDemand-2018-0001

Summary:

When submitting a job, the user can craft a task that continuously spawns new processes and consumes all available resources on the executing node. This can result in a Denial of Service (DoS) on the executing node.

Component	Vulnerable Versions	Platform	Availability	Fix Available
Open OnDemand	All	Non-Windows	Not known to be publicly available	No
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified on virtual machine testbed	Local ordinary user with Open OnDemand privileges	Internet connected submit machine	Low	Medium
Fixed Date	**Credit**			
n/a	Elisa Heymann Joe Atkins			

Access Required: Local ordinary user with Open OnDemand submission privileges.

This vulnerability requires the user to be able to submit jobs to an Open OnDemand portal.

Effort Required: Low

Exploiting this vulnerability requires the user to submit a job with a script which calls a malicious program.

Impact/Consequences: Medium

If this vulnerability is exploited, the consequences include a denial of service on the executing node.

Full Details:

This exploit is executed by submitting malicious code to be run on the executing node. As Open OnDemand does not conduct input validation it can be used to submit malicious code to be executed at the site.

To create a denial of service on the executing node, the attacker can entire and compile the C code shown in Figure 1, which when called creates continuous forking of new processes. The attacker can then submit the shell script shown in Figure 2 which executes the C progam on the executing node.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
        int i;
        printf ("In C\n");
        for (i=0; i<1000000; i++) {
                if (fork() != 0)
                        printf ("*\n");
        }
}
```

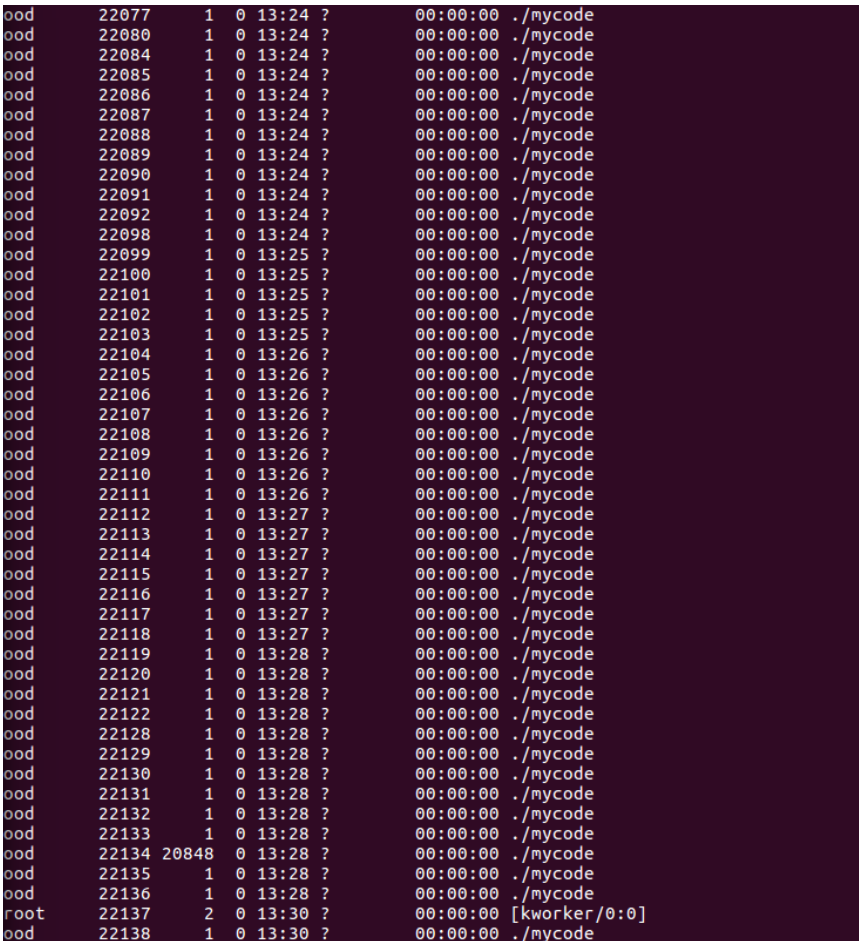
Fig1. - Malicious C program containing code resulting in continuous forking of new processes.

```
#!/bin/bash

./mycode
echo "In Shell"
```

Fig2. - Bash script calling malicious program `mycode` from fig 1.

When the above code is processed by the executing node, the shell script executes the program `./mycode`. This execution starts a continuous forking of the process. spawning a new copy of itself, eventually overloading the system.



```
ood      22077     1  0 13:24 ?        00:00:00 ./mycode
ood      22080     1  0 13:24 ?        00:00:00 ./mycode
ood      22084     1  0 13:24 ?        00:00:00 ./mycode
ood      22085     1  0 13:24 ?        00:00:00 ./mycode
ood      22086     1  0 13:24 ?        00:00:00 ./mycode
ood      22087     1  0 13:24 ?        00:00:00 ./mycode
ood      22088     1  0 13:24 ?        00:00:00 ./mycode
ood      22089     1  0 13:24 ?        00:00:00 ./mycode
ood      22090     1  0 13:24 ?        00:00:00 ./mycode
ood      22091     1  0 13:24 ?        00:00:00 ./mycode
ood      22092     1  0 13:24 ?        00:00:00 ./mycode
ood      22098     1  0 13:24 ?        00:00:00 ./mycode
ood      22099     1  0 13:25 ?        00:00:00 ./mycode
ood      22100     1  0 13:25 ?        00:00:00 ./mycode
ood      22101     1  0 13:25 ?        00:00:00 ./mycode
ood      22102     1  0 13:25 ?        00:00:00 ./mycode
ood      22103     1  0 13:25 ?        00:00:00 ./mycode
ood      22104     1  0 13:26 ?        00:00:00 ./mycode
ood      22105     1  0 13:26 ?        00:00:00 ./mycode
ood      22106     1  0 13:26 ?        00:00:00 ./mycode
ood      22107     1  0 13:26 ?        00:00:00 ./mycode
ood      22108     1  0 13:26 ?        00:00:00 ./mycode
ood      22109     1  0 13:26 ?        00:00:00 ./mycode
ood      22110     1  0 13:26 ?        00:00:00 ./mycode
ood      22111     1  0 13:26 ?        00:00:00 ./mycode
ood      22112     1  0 13:27 ?        00:00:00 ./mycode
ood      22113     1  0 13:27 ?        00:00:00 ./mycode
ood      22114     1  0 13:27 ?        00:00:00 ./mycode
ood      22115     1  0 13:27 ?        00:00:00 ./mycode
ood      22116     1  0 13:27 ?        00:00:00 ./mycode
ood      22117     1  0 13:27 ?        00:00:00 ./mycode
ood      22118     1  0 13:27 ?        00:00:00 ./mycode
ood      22119     1  0 13:28 ?        00:00:00 ./mycode
ood      22120     1  0 13:28 ?        00:00:00 ./mycode
ood      22121     1  0 13:28 ?        00:00:00 ./mycode
ood      22122     1  0 13:28 ?        00:00:00 ./mycode
ood      22128     1  0 13:28 ?        00:00:00 ./mycode
ood      22129     1  0 13:28 ?        00:00:00 ./mycode
ood      22130     1  0 13:28 ?        00:00:00 ./mycode
ood      22131     1  0 13:28 ?        00:00:00 ./mycode
ood      22132     1  0 13:28 ?        00:00:00 ./mycode
ood      22133     1  0 13:28 ?        00:00:00 ./mycode
ood      22134 20848  0 13:28 ?        00:00:00 ./mycode
ood      22135     1  0 13:28 ?        00:00:00 ./mycode
ood      22136     1  0 13:28 ?        00:00:00 ./mycode
root     22137     2  0 13:30 ?        00:00:00 [kworker/0:0]
ood      22138     1  0 13:30 ?        00:00:00 ./mycode
```

Fig2. - ps on executing node showing infinite recursion of malicious program

The screenshot of the output of the ps command (Figure 2) demonstrates that this attack can continuously fork child processes that will consume all execute node system resources.

Cause: Lack of input validation files and scripts supplied by user when submitting jobs.

As there is no input validation performed the user can insert maliciously crafted code into submitted files and scripts resulting in attacks on the executing host.

Proposed Fix:

The proposed fix is to implement input validation or resource limitations in the configuration for the batch environment (e.g. SLURM, TORQUE).

Actual Fix:

n/a.

Acknowledgment:

This work is supported in part by the NSF Cybersecurity Center of Excellence under National Science Foundation Cyber Infrastructure grant ACI-1547272.