Open XDMoD First Principles Vulnerability Assessment

August 7, 2020

Draft Final Report

Distribution: Release to public after February 1, 2021

Ian Ruh[1], Elisa Heymann[2], Barton P. Miller[3]

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

About Trusted CI

The mission of Trusted CI is to provide the NSF community with a coherent understanding of cybersecurity, its importance to computational science, and what is needed to achieve and maintain an appropriate cybersecurity program[4].

Acknowledgments

Trusted CI's engagements are inherently collaborative; the authors would like to thank the XDMoD team, specifically Ryan Rathsam, for the collaborative effort that made this document possible. The authors would also like to thank Ben Kinzer for his involvement in the initial steps of this assessment.

This document is a product of the Center for Trustworthy Scientific Cyberinfrastructure (Trusted CI). Trusted CI is supported by the National Science Foundation under grant 1920430. For more information about the Center for Trustworthy Scientific Cyberinfrastructure please visit: https://trustedci.org/. Any opinions, findings, and 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:

Ian Ruh, Elisa Heymann and Barton P. Miller. "Open XDMoD First Principles Vulnerability Assessment". TrustedCI: The NSF Cybersecurity Center of Excellence. August 2020.

This work is available on the web at the following URL:
http://hdl.handle.net/2142/107785

[4] https://trustedci.org/mission

Table of Contents

List of Figures

Executive Summary

Trusted CI assessed the security of Open XDMoD, an open source tool for the management of high performance computing (HPC) resources, in collaboration with the XDMoD team. Open XDMoD provides a web portal for graphical analysis of the performance of HPC environments, and provides the backend infrastructure for importing and processing of accounting logs from local resource management systems (LRMS).

We conducted an in-depth vulnerability assessment of Open XDMoD by applying the First Principles Vulnerability Assessment (FPVA)[5] methodology. The FPVA analysis starts by mapping out the architecture and resources of the system, paying attention to the trust and privileges of components, and identifying the high value assets. From there we perform a detailed inspection of the parts of the code that have access to the high value assets.

We assessed Open XDMoD using a Docker container provided by the Open XDMoD team and a CentOS 7 VM with Open XDMoD installed. Specifically, we assessed the 9.0 version branch of the Open XDMoD repository as it was on March 27, 2020, but the branch continued to be developed while the Open XDMoD team worked towards the 9.0.0 release. The assessment covered the core code for the REST API, including the code for authentication, authorization, database access and database modification, and for the importing and processing of accounting log files. We collected the results from each step of applying the FPVA methodology, generated vulnerability reports and delivered them to the XDMoD team, and released this final report to the Open XDMoD team at the end of the engagement. Open XDMoD also includes an optional module to incorporate SUPReMM, a job performance monitoring tool that is also developed by the Open XDMoD team. Due to the length of this engagement, this module was not explored for this assessment.

This report includes a discussion of the vulnerabilities found, several non-security related bugs, and the parts of Open XDMoD that were inspected where no apparent issues were found. The parts where no issues were found included the majority of the code for performing SQL queries to the data warehouse, checking the authorization of requests from the REST API, user authentication, and sanitizing user input. Though it is impossible to certify that code is free of vulnerabilities, we have substantially increased our confidence in the security of those parts of the code.

We found three security vulnerabilities during our assessment. The first vulnerability is that use of HTTPS over HTTP is not mandatory (though it is suggested in the documentation).

[5] 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.

Communicating over unencrypted channels exposes all information sent between the client and the server, including cookies and passwords. Second, Open XDMoD was found to be logging sensitive information to a globally readable file. Third, a DoS attack was possible by filling all the free space on a disk partition containing Open XDMoD's log files, causing the web portal to become inoperable. Additionally, the assessment found that OpenXDMoD relies on three software dependencies that are either unmaintained or out of date. All the security issues found, as well as non-security related bugs, are detailed in Section 6.

1 Overview

This document describes the engagement between Trusted CI and Open XDMoD that occurred from March to August 2020. The goals of the engagement were to evaluate the technology and architecture of Open XDMoD and perform a code-level security review of the Open XDMoD software.

1.1 Background

Open XDMoD is an open source system for monitoring the performance and utilization metrics of HPC systems. Open XDMoD provides a web application that is accessible to individual users and to institutional managers to analyze and explore metrics via interactive charting and querying. Without dependencies, Open XDMoD is approximately 130,000 lines of code, of which 72,000 are PHP, and 44,000 are JavaScript. The remaining code is a mixture of JSON, Markdown, CSS, HTML, SQL, Java, and other utility languages or formats.

1.2 Methodology

This engagement focused on performing FPVA on Open XDMoD. The engagement for Open XDMoD started on March 27, 2020 when we received a Docker container from the Open XDMoD team. We assessed the 9.0 version branch of the Open XDMoD repository as it was on March 27, 2020, but the branch continued to be developed while the Open XDMoD team worked towards the 9.0.0 release.

We used both the docker container provided by the Open XDMoD team and a CentOS 7 VM installed with Open XDMoD. The docker container was used during the creation of the architectural diagrams. However, as the docker container was configured such that many processes ran with higher permissions than in a normal install, to perform the trust and privilege analysis step of the FPVA process (detailed in Section 2), we created a minimal CentOS 7 VM installed with Open XDMoD, according to the installation instructions in the documentation, that better reflected a production environment.

Due to limited time and resources, our FPVA assessment did not include the optional SUPReMM module, which is also developed by the Open XDMoD team.

2 Overview of First Principles Vulnerability Assessment

First Principles Vulnerability Assessment (FPVA) is an analyst-centric (manual) methodology that aims to focus the analyst's attention on the part 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:** determined the major structural components of the system and how they interact. At this point, we produced Architectural Diagrams that illustrate the structure of the system. The primary deliverables of this step were the processes and hosts, and their interactions, as shown Figures 1, 2, 3, and 4.
2. **Resource Identification:** identified key resources accessed by each component. Examples of these resources include files, databases, logs, and devices. The Resource Diagrams that we produced illustrate these resources and their connection to system components. The primary deliverable of this step was the addition of the resources shown in Figures 1, 2, 3, and 4.
3. **Trust and Privilege Analysis:** identified 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:** examined relevant components in depth. A key aspect of the FPVA methodology 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 was logged for inclusion in the final report.
5. **Dissemination of Results:** we created a report for each vulnerability found that was then delivered to the development team as it was completed. We prepared a final report that includes the deliverables mentioned above as well as an outline of the work completed. We include identified bugs as well as areas that have been investigated but where no bugs or vulnerabilities were found. We then disseminated the final report to the requesting parties (i.e., the lead of the development team).

We note that Open XDMoD is a large software system, so no time-limited 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 XDMoD relies on a variety of software dependencies. As such, there should be ongoing attention to the security of the external software that Open XDMoD relies on.

3 Architectural Analysis

Based on our study of the Open XDMoD documentation[6], testing environment, and code, we identified the attack surfaces, core processes, and hosts. Figure 1 shows the overview of the Open XDMoD architecture. Figures 2, 3, and 4 show the detailed architectural diagrams.

Figure 1 shows the three groups of processes that perform the core function of Open XDMoD. The first group consists of the processes created by Apache to handle each request to the REST API. The second group consists of the shredder and ingestor processes that import the accounting logs and process the logs, respectively. These processes can be launched manually from a shell or, in a normal production configuration, from cron. The third group consists of background processes that perform periodic tasks including generating reports and exporting data.

The accounting log files from multiple resource managers can be processed by Open XDMoD, however, the mechanism for processing the accounting logs from Slurm differs from that of processing the accounting logs of other resource managers and is therefore separated into its own diagram. We elaborate on our findings in the following subsections.

[6] https://open.xdmod.org/8.5/

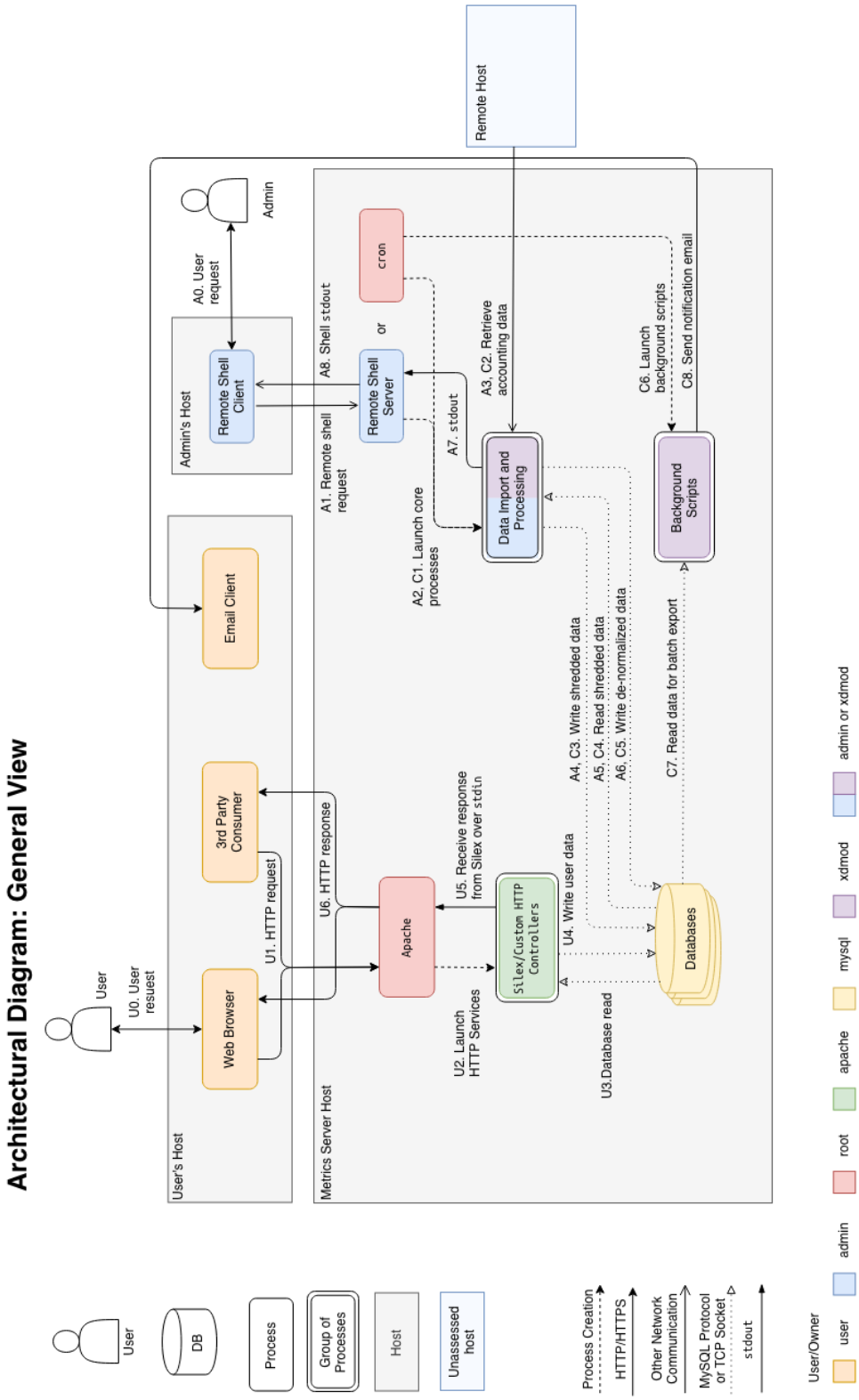

Figure 1. Architectural Diagram: General View

3.1 Attack Surface

From the Architectural Diagrams, we identified the following points on the attack surface:

- **Open XDMoD Web Portal:** the interface used by both regular users and administrative users for accessing the accounting data. Through the web portal they can make queries, export data, and create reports.
- **Open XDMoD REST API:** facilitates the web application's access to accounting log data, and allows for the manipulation of access control lists (ACLs).
- **Open XDMoD log files:** Open XDMoD and its dependencies write to several log files in `/var/log/xdmod/`. The log files can be written to by both Apache and the Open XDMoD REST API, as seen in Figure 2.
- **Accounting log files:** files from the LRMS are processed and stored in the data warehouse to be accessed from the REST API. The accounting log files are shown in Figures 3 and 4.
- **Command line interface:** perform specific tasks that affect Open XDMoD, such as shredding and ingesting log files.
- **Configuration files:** Open XDMoD uses a large number of configuration files that specify application parameters such as user host, database host, port and password, and define the extract, transform, and load (ETL) pipeline that processes the accounting logs. The configuration files can be seen in Figures 2, 3, and 4.

3.2 REST API

Open XDMoD uses two different methods for handling requests to the REST API, as shown in Figure 2. Open XDMoD relies on the PHP framework Silex[7] for part of the REST API and uses custom PHP scripts for the other endpoints. Each endpoint consists of the requested path and the HTTP method associated with the request. The Apache web server receives incoming HTTP requests and routes them to the appropriate PHP file. It launches a new process for every request, resulting in either an instance of a Silex application or an instance of a custom REST controller, depending on the requested route. The controller handles the routing and authentication for each request. The Silex application receives the requests made to `rest/` routes, whereas the custom REST controllers are used when a request is made to one of the PHP files in `html/controllers/`. Details of these two routes are elaborated in Section 3.2.1.

[7] https://silex.symfony.com/

Both Open XDMoD and Apache are configured to log to files in the `/var/log/xdmod` directory. Entries to the files are made in exceptional circumstances to log error messages and in the normal operation of the system. Open XDMoD uses the `logrotate`[8] utility to manage log files over time. The function and configuration of `logrotate` is detailed in Section 3.2.2.

Each of the endpoints for the REST API perform a different operation. An endpoint may query or modify entries in the data warehouse or it may launch a new process. Users are able to create reports that are populated with data from the data warehouse. They can configure reports to be sent out periodically by email or they can create one time reports to download. Open XDMoD uses the JasperReports Java library to create the PDF and Word files of the reports. When a user requests a report, the server first launches PhantomJS to construct the images of the charts, then launches the ReportBuilder Java program to create either a PDF or a Word version of the report. The PhantomJS and ReportBuilder operations are also performed in the background when a report is configured to be sent periodically. In that case, the processes are launched by the Report Manager background script, which is explained in Section 3.2.3.

[8] https://github.com/logrotate/logrotate

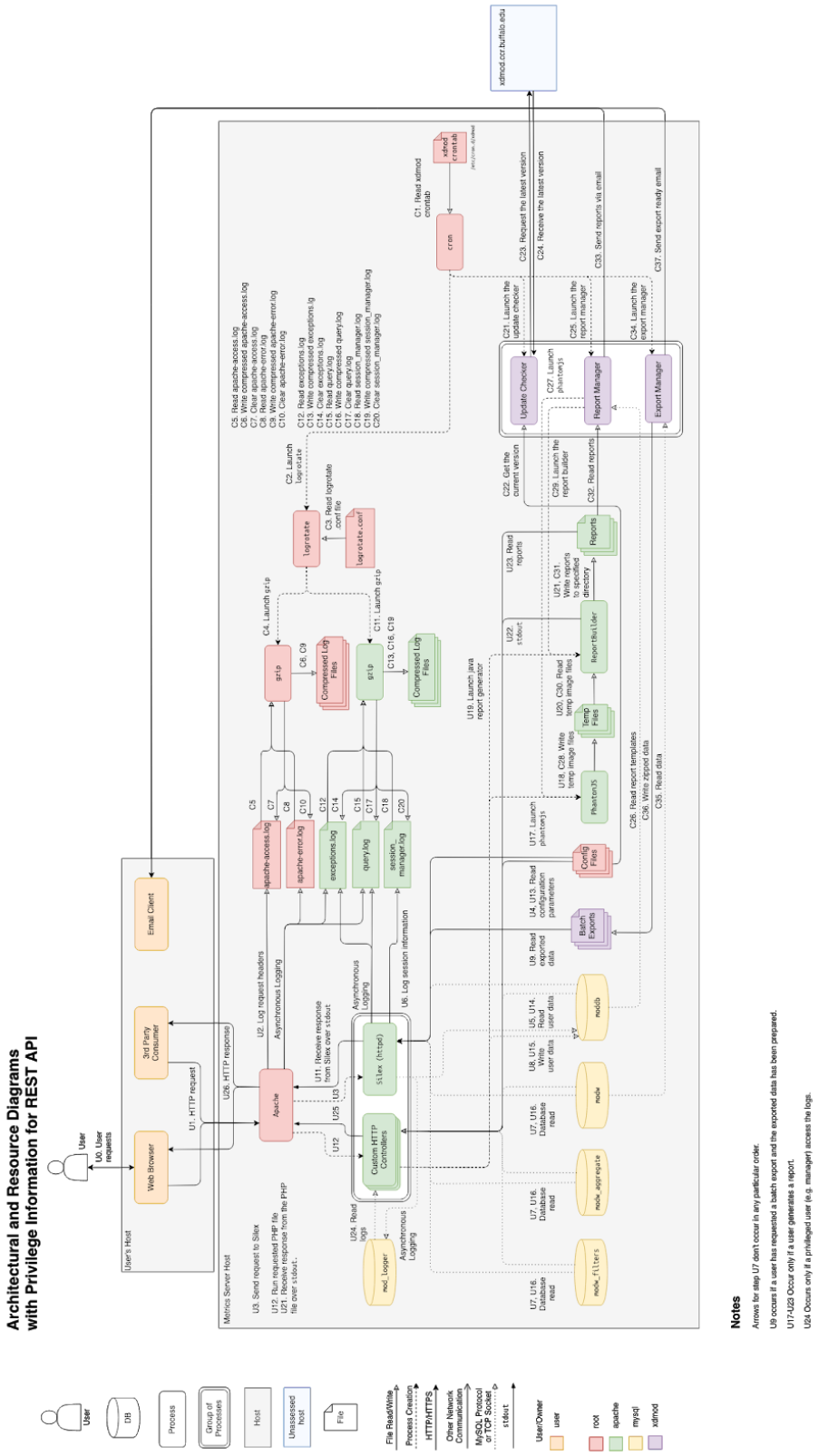

Figure 2. Architectural and Resource Diagram with Privilege Information for the REST API

3.2.1 Parallel REST APIs

Open XDMoD uses two mechanisms for providing the REST API that is used by the Open XDMoD web portal, as shown in Figure 2. The primary mechanism is through the Silex HTTP framework, which handles routing of requests to registered request handlers. The other mechanism, based on our conversations with the developers, is a legacy API that is being transitioned to Silex and relies on Apache to route requests to specific PHP scripts.

Both APIs provide endpoints for user authentication (login and logout), though only the Silex endpoints are used in the Open XDMoD web portal. Other functionality for the web portal is split between the two mechanisms. All the operations for the internal dashboard (a dashboard available only to managers that allows user management, account creation, and system management) are performed using the custom HTTP controllers. An outline of the partitioning between the mechanisms is shown below.

Silex

- Query the data warehouse.
- Initiate and download a batch export.
- Manage queries for the metric explorer.
- Manage dashboard layout.

Custom HTTP Controllers

- Update user password and profile.
- Request reset password email.
- Manage available filters for the metric explorer.
- Send email messages on behalf of users from interactions with the “Contact” and “Sign Up” web portal features.
- Manage the set of available charts for use in a report.
- Create and edit reports to be sent periodically.

Both

- User login and logout.

3.2.2 Open XDMoD Log Management

There are five log files used in the `/var/log/xdmod` directory. Of those, four are written to by Apache, and three by the PHP request handlers.

The `logrotate` utility is used to manage the growth of the logs over time. When `logrotate` is installed, a crontab entry is added `/etc/cron.daily/logrotate` to run `logrotate` daily by default. When logrotate launches, it reads the

`/etc/logrotate.d/xdmod` file that defines the operations to be performed on the Open XDMoD log files. All the files are compressed by gzip and renamed with the date for storage.

3.2.3 Background Scripts

Open XDMoD comes with three background scripts configured to run periodically, the update checker, report manager, and export manager scripts. All three are launched by cron at the times defined in the `/etc/cron.d/xdmod` file.

The update checker background script runs monthly by default and checks the version of Open XDMoD running locally to compare it against the "latest version". The University at Buffalo Center for Computational Research (the Open XDMoD development team) Open XDMoD instance is used as the source of truth in determining what the "latest version" is. The script makes a request to https://xdmod.ccr.buffalo.edu/rest/v0.1/versions/current to retrieve the latest version.

The report manager script is run by default at 03:00 daily. It determines the reports that are configured to be sent out that day by checking the `moddb.Reports` table. It then queries the data warehouse for the requisite data. Similar to the operations performed when a report is generated at the request of a user using the REST API, PhantomJS and the ReportBuilder programs are launched to construct the charts and the PDF or Word file. When the report has been constructed, the report manager sends the report as an email to the configured user(s).

When a user requests a batch export of data from the data warehouse, the data is not immediately gathered. Their request is added to a table and is handled by the export manager script at a later time. The export manager script is configured by default to run at 04:00 daily. It checks the `moddb.batch_export_requests` table for any outstanding requests. For each request, it retrieves the data from the data warehouse, compresses the data to a zip file, and writes the file to the configured export directory, which is `/var/spool/xdmod/export` by default. The user that requested the export is sent an email notifying them that the export is ready for download.

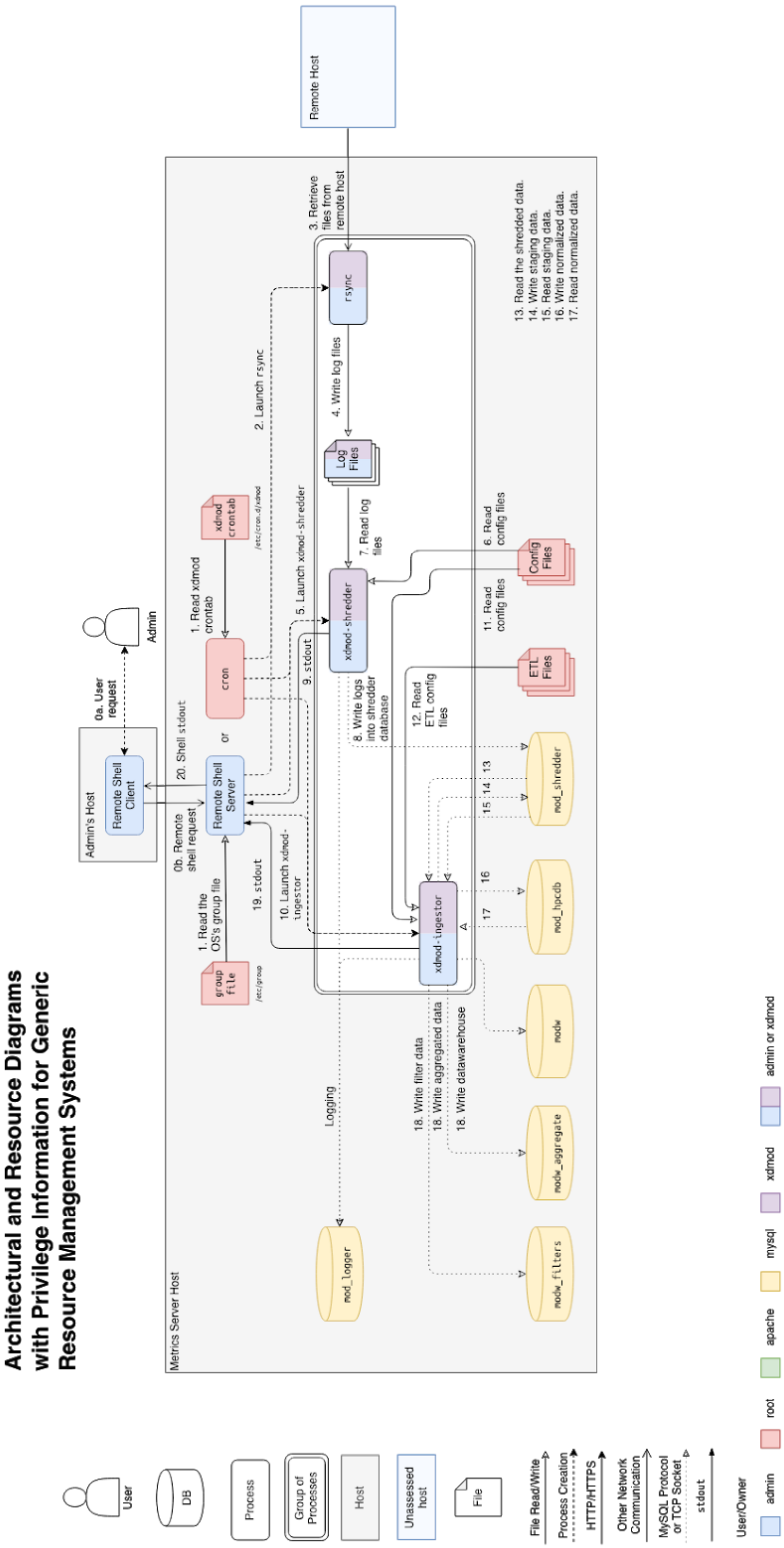

Figure 3. Architectural and Resource Diagram with Privilege Information for Generic Resource Management Systems

3.4 Generic Resource Manager Log Processing

Figure 3 shows the processing of log files from a generic resource manager and the loading of the information into the data warehouse. The shredder and ingestor processes can be initiated by either cron or by a system administrator manually executing the scripts from a shell on the host system. The blue and purple backgrounds in Figure 3 indicate that either a system administrator or the xdmod user (used by cron when running the processes) owns the process or file. In either case, the xdmod-shredder script is run to process the accounting logs from the resource manager.

Once the shredder has finished loading the logs into the mod_shredder database, the xdmod-ingestor script is run to move the shredded logs through the data pipeline into the data warehouse.

When configuring Open XDMoD, it is left up to the system administrator to determine the mechanism that will get the accounting log files from the local resource management system node to the metrics server host. Though not explicitly specified in the documentation, based on communication with the developers, this normally happens one of two ways: using a cron job configured to run rsync to transfer files from a remote host periodically, or accessing the accounting log files through a shared file system.

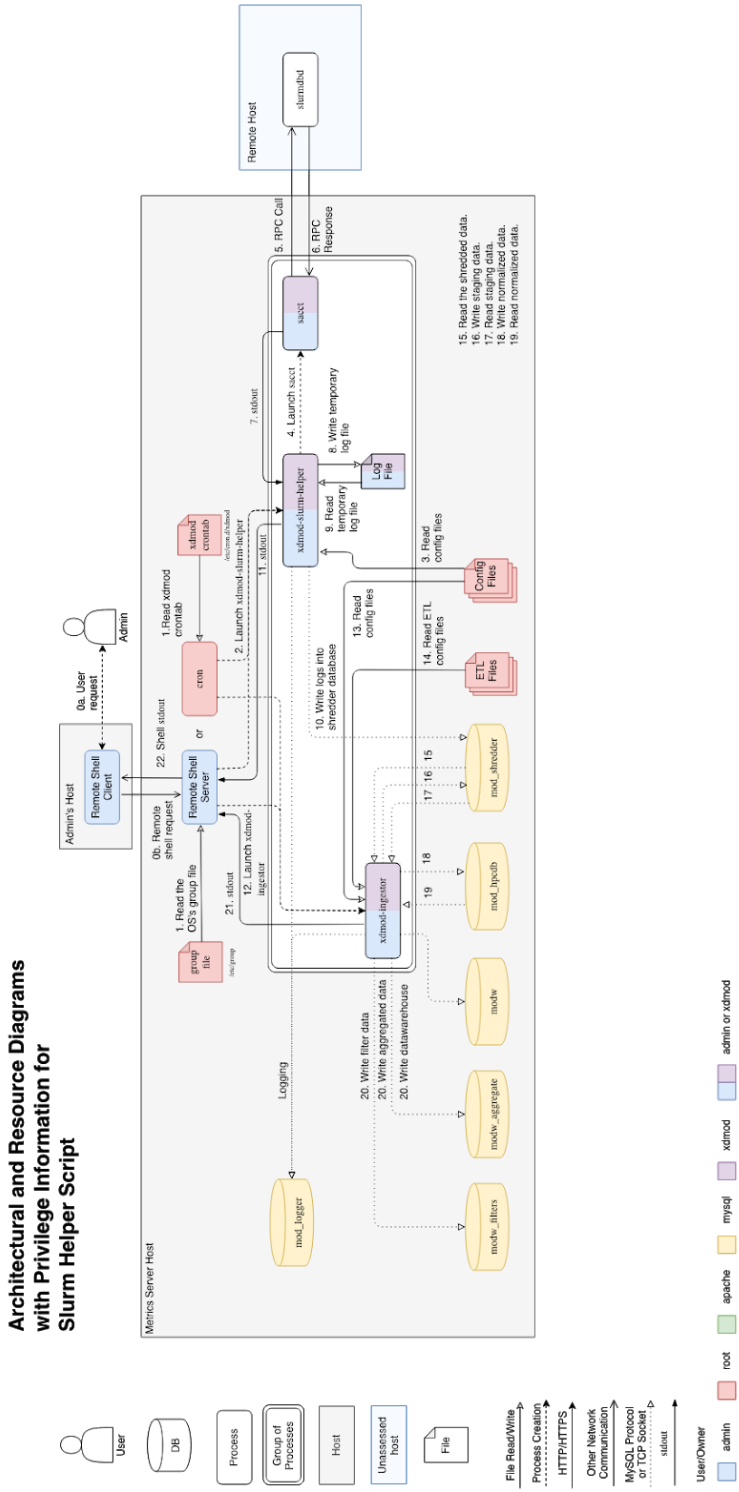

Figure 4. Architectural and Resource Diagram with Privilege Information for the Slurm Resource Management System

3.5 Slurm Log Processing

Figure 4 shows the processing of accounting logs from the Slurm resource management system. The process follows a similar path to a generic resource management system. However, in the mechanism for the retrieval of the logs, rather than directly processing a log file, the xdmod-slurm-helper script uses the sacct (Slurm Accounting) utility provided by Slurm to retrieve the accounting logs from the cluster and then write the data to a temporary file. That file is then processed through the same mechanisms as a generic resource manager's log files.

4 Resource Identification

From the architectural and resource diagrams in Figures 2, 3, and 4, we were able to identify critical resources for further investigation, including configuration files, accounting log files, Open XDMoD log files, temporary files, and databases. Each of these is elaborated in the sections below.

4.1 Configuration Files

Open XDMoD has around 500 configuration files, most of which are in the `/etc/xdmod/` directory, with additional configuration files for cron and logrotate in `/etc/cron.d/` and `/etc/logrotate.d/` respectively. The primary config file is `/etc/xdmod/portal_settings.ini`, which defines the web portal's base URL, the name of log files, the host, port, user name, and password for all the databases, and other settings. The ETL pipeline for processing accounting logs is defined by a set of JSON files stored in the `/etc/xdmod/etl/` directory.

All the config files, including the ones in `/etc/xdmod/`, `/etc/cron.d/`, and `/etc/logrotate.d/`, are owned by root and have permissions of `644/-rw-r--r--`, except the `portal_settings.ini` file which has permissions `0440/-r--r-----`.

4.2 Accounting Log Files

The accounting log files are one of the primary resources in the system. When a user of a cluster submits a job to the LRMS, a log of the job and its properties is created. Open XDMoD uses the logs generated by the LRMS to gather the accounting information stored in the data warehouse and shown in the Open XDMoD web portal.

Depending on the LRMS being used, there are two different mechanisms that are used to get the accounting logs into the data warehouse, as shown in Figures 3 and 4. For a generic LRMS,

the process by which the accounting log files are moved from the cluster to the Open XDMoD metrics server is not specified. Based on our communications with the Open XDMoD development team, a normal installation may use rsync to transfer between the hosts on a periodic basis, or use a shared file system between the cluster and the metrics server. In both cases, it is left up to the user to ensure that the files are moved securely.

If Slurm is the LRMS being used by the cluster, then Open XDMoD provides the `xdmod-slurm-helper` script to facilitate retrieval of the accounting logs. The helper script uses the `sacct` utility to retrieve the logs from the cluster and writes them to a temporary file. This file is immediately ingested and follows the same process as the generic LRMS log processing.

4.3 Open XDMoD Logs

Open XDMoD and Apache write to five log files in the `/var/log/xdmod/` directory. Log entries may be written in during exceptional circumstances or during the normal operation of the system. The files and their contents are outlined below.

- `apache-access.log`: every time a request is made to Apache, an entry is logged containing the timestamp, HTTP method, route, and client information. This file is only written to by Apache. This log was one of the files used to create a DoS attack, detailed in Appendix C.
- `apache-error.log`: this file is written to when a request causes Apache to throw an error. This file is also one of the files used to create a DoS attack, detailed in Appendix C.
- `exceptions.log`: this file is written to when unhandled exceptions are caught at the top level of the Open XDMoD PHP stack.
- `query.log`: this file is used to log queries made to the data warehouse.
- `session_manager.log`: every time a request is made to the Open XDMoD REST API, the metrics server attempts to verify the `xdmod_token` cookie belongs to a valid session. When it queries the `moddb` database, it also logs the query to this file. This was the source of the vulnerability detailed in Appendix B.

4.4 Temporary Files

Open XDMoD writes temporary files to the system's temporary directory, as determined by the PHP `sys_get_temp_dir` function. When Open XDMoD builds a report to be sent out by email, or downloaded through the web portal, the charts are generated as images by PhantomJS, which are stored as files in the temporary directory. Additionally, a temporary file

containing the accounting logs output by the sacct utility is written to the system's temporary directory when the Slurm helper script is executed.

4.5 Databases

Open XDMoD uses multiple databases for storing the accounting logs in the data warehouse, for storing user information, and for storing log messages. The databases can be seen in Figures 2, 3, and 4, and the purpose of each is described below.

- `mod_shredder`: stores data that has been retrieved from the resource managers but not yet normalized.
- `mod_hpcdb`: stores normalized data that has not yet been loaded into the data warehouse.
- `modw`: data warehouse.
- `modw_aggregate`: stores aggregated data from the data warehouse to speed up response times for certain queries.
- `modw_filters`: stores filtered data from the data warehouse to speed up response times for certain queries.
- `mod_logger`: stores log messages from the data shredding and ingestion processes.
- `moddb`: stores user data, session keys, some configuration information.

5 Trust and Privilege Analysis

There is implicit trust in any process running as `root` or any resource owned by `root`. Any communication from an unprivileged process to a privileged process needs to be investigated, and any data read in from a resource needs to be validated.

5.1 REST API

Open XDMoD has five default authorization levels that can be assigned to registered users, and one special public user level that is assigned to any unregistered user. These levels, in order of increasing permissions, are public, user, principal investigator, center staff, center director, and manager.

When a user or a third party makes an HTTP request to the Open XDMoD REST API, the request is validated using the `xdmod_token` and `PHPSESSID` cookies. If these cookies are not present or cannot be verified then the user is assigned the public user role. Open XDMoD uses the cookies to authenticate the user making the request, and from that retrieves the data they are authorized to access and the actions they are permitted to perform.

5.2 Accounting Log Processing

The Open XDMoD shredding and ingesting processes are created in two ways. By default, the `/etc/cron.d/xdmod` file is configured so the processes are run by cron using the `xdmod` user ID (shown in purple in Figures 1 through 4). However, Open XDMoD also allows any user ID belonging to the `xdmod` group to run the background processes (shown in blue in Figures 1, 3, and 4).

The ability for a user to run either the shredder or ingestor processes is dependent on the ability of the user to read the necessary configuration files, not on the permission bits associated with the executables. All the background scripts' permission bits are `0755/-rwxr-xr-x`, so any user can run the executables. However, the programs fail when they attempt to read the configuration files that store the database connection metadata (host, port, user, password), unless they are in the `xdmod` group.

5.3 Access Privileges

We note that there are two different mechanisms for controlling access to the Open XDMoD portal and its resources: the user authorization provided by the REST API described in Section 5.1, and the authorization for running the scripts described in Section 5.2. The authorization control provided by the REST API is managed through the Open XDMoD web portal by a manager, whereas the authorization to run the scripts is dependent on the user ID's membership to the `xdmod` Linux group (based on editing the contents of the system configuration file `/etc/group`).

Similarly, there are two different mechanisms for authenticating with Open XDMoD. The user accounts stored and managed by the REST API provide access via the web portal, and the Linux user ID's managed by the operating system provide access to the Open XDMoD background scripts and backend resources.

6 Component Evaluation

This section describes some of the areas of focus for the component analysis step of our assessment. In this step, we performed code inspection looking for weaknesses that could be exploited.

6.1 Vulnerabilities Found

6.1.1 HTTP vs. HTTPS

Summary

HTTPS should always be used over HTTP. The associated vulnerability report is in Appendix A.

Description

The default configuration of Open XDMoD does not encrypt HTTP traffic. This allows attackers to monitor all traffic between the server and the client. As a result, passwords submitted on login are sent in plain text, and the `xdmod_token` cookie can be stolen from any HTTP request after authentication.

Mitigation

This can be mitigated by enforcing HTTPS on all traffic to/from the server. Apache should be configured by default to allow only HTTPS and provide a self- signed certificate until the user replaces it with their own. Also ensure that all cookies are sent with the "secure" flag. This flag prevents cookies from being sent over unencrypted channels.

6.1.2 Insecure Log Files

Summary

The XDMoD software logs the session cookies for every request it receives in the process of checking their validity. The file being logged to is globally readable and could allow an attacker to hijack active sessions if they have an account on the metrics server host. The associated vulnerability report is in Appendix B.

Description

Every time a request is made to the Open XDMoD API, the metrics server attempts to verify the `xdmod_token` cookie belongs to a valid session. In querying the moddb database, the query, and the parameters of the query (including the cookies being checked), are logged to the `/var/log/xdmod/session_manager.log` file. This log file is globally readable. By reading the log file an attacker can see any active session cookies and therefore be able to impersonate any of the active sessions.

Mitigation

The log file can be secured by just changing the file permissions from the globally readable `0644/-rw-r--r--` to the only group and owner readable `0640/-rw-r-----`.

6.1.3 Denial of Service Vulnerability

Summary

Open XDMoD is vulnerable to a Denial of Service (DoS) attack that prevents proper functioning of the Open XDMoD portal. The associated vulnerability report is in Appendix C.

Description

Every time a request is made to Apache, an entry is logged to `/var/log/xdmod/apache-access.log` containing the timestamp, HTTP method, route, and client information. By repeatedly sending requests (on any route) to Apache, an attacker can fill up the free space on the file system partition. This results in the XDMoD web portal failing to properly load due to the malformed page returned by the server. In addition to XDMoD, any other programs that use the same file system partition as the log files may also be affected.

Mitigation

One option to fix this vulnerability is to use the Apache feature of piped logs and the Apache-provided `rotatelogs` utility, which allows continuous monitoring and control of file size.

An additional mitigation is to isolate log files to their own file system partition. In the case that these files grow to a large size, this will isolate the effect and reduce the impact. This approach can be used with any other approach to add an additional layer of protection.

6.2 Items to Improve

6.2.1 Outdated and Vulnerable Dependencies

Summary

The current version of Open XDMoD has three problematic dependencies. Two of the dependencies are unmaintained and the third is outdated and vulnerable (though it is not clear if it is exploitable in Open XDMoD). The associated warning report is in Appendix D.

Description

Open XDMoD uses the no-longer-maintained HTTP framework, Silex, that had an end-of-life in June of 2018. Although there are currently no publicly known vulnerabilities in Silex, the lack of active development and maintenance increases the likelihood that vulnerabilities will not be caught and will not be fixed. Open XDMoD also relies on PhantomJS, development of which has been suspended, and which has a publicly known vulnerability. Additionally, Open XDMoD uses an outdated and vulnerable version of the JasperReports Library Community Edition (where the vulnerability reports are currently too terse to tell if they affect Open XDMoD).

Mitigation

Using actively maintained software lowers the risk of vulnerabilities going uncaught and unfixed. In the case of Silex, the recommendation from the developers is to migrate to using Symphony and Flex, which continue to be actively maintained and developed. As PhantomJS is no longer maintained, it is recommended that an alternative be found with equivalent functionality.

Keeping all libraries and frameworks updated to their most recent versions prevents numerous vulnerabilities. Jaspersoft has fixed the known vulnerabilities in newer versions of JasperReports Library CE.

6.2.2 Insecure Code Practice

Summary

XDMoD uses certain coding practices that, while not known to be exploitable at the moment, are insecure and should be avoided.

Description

When authenticating a request, either from the session cookies or a provided user name and password, XDMoD performs a query to select all entries from the database that match the given credentials. This operation returns a list of entries, not necessarily one. In both checking the session cookies and verifying the user's password, the only verification of the number of entries done is whether the list is empty or has greater than 0 entries. If it has at least one entry, then the first entry is assumed to be the intended entry.

This practice can be problematic if an attacker is able to corrupt the query to return multiple entries in the table, rather than just one. Such a corruption can be detected by verifying that exactly one entry matches the query and failing if multiple entries match.

Location

This issue is present in two places: where the user name and password are checked in the `XDUser::authenticate()` method and where the session cookies are checked in the `Authentication::resolveUserFromToken()` method.

6.2.3 Report Download REGEX Handling

Summary

The `html/controllers/report_builder.php` endpoint contains a `download_report` operation that requires two additional parameters: a `report_loc` parameter and a `format` parameter. The two parameters are used to construct a path to the report file, which is read and sent back to the client. Both parameters are validated using a

regular expression. However, when the validation fails, the operation defaults to attempting to read one of two specific files, if they exist.

Description

The validation using a REGEX protects against arbitrary directory traversal attacks. However, when the REGEX validation fails for the `report_loc` parameter, the code defaults to attempting to read either the `/tmp/.pdf` or `/tmp/.doc` file depending on the format specified. These files are not expected to exist though, nor, if they do, are they likely to contain valuable information. However, if they do exist, and their permission bits allow the Apache process to read them, they can be downloaded from the server. It is suggested that the failure case of such a validation for user input not be a default value, but instead throw an exception or be explicitly handled.

Location

The problem exists in the operation handler `html/controllers/report_builder/download_report.php`.

6.2.4 Bug in User Profile Updating

Summary

Open XDMoD allows a user to update several fields (including first name, last name, email, and password) from the "Profile" popup in the Open XDMoD web portal. While there is client side validation of the fields occurring before the requests are sent, there is no server-side validation on the first name, last name, or email fields. Since the DB schema restricts the length of these fields by using the varchar([num]) datatype, the lack of server-side validation forces MariaDB to truncate the values. This can lead to malformed email addresses.

Description

There are two places where user profile information can be set. The first is by an administrator in the internal dashboard, the second is by the user themselves. When an administrator sets the profile information for another user, they use the `/controllers/user_admin.php` endpoint with the operation set to `update_user`. When a request is received, all the fields, except email, are verified to conform to a set of regular expressions. The email field, however, does not get checked. The email column in the database schema for moddb.Users is varchar(200). If the email in the request is longer than 200 characters, MariaDB truncates the email when the entry is inserted or updated.

A similar situation can occur with the endpoint used when a user updates their profile information. The Open XDMoD webportal performs validations of their input, but the server does not. Since the columns in the moddb.Users database for first name, last name, and email

are varchar(50), varchar(50), and varchar(200) respectively, the values are truncated when updated in the database.

While the first name and last name were not found to have any functional impact on the system, the truncation of the email can prevent any communication to the owner of the account (including for maintenance announcements or password resets).

In addition to truncation of certain fields, the endpoint for users to update their profile information (PATCH: `/rest/v1/users/current`), also due to the lack of server side validation, can cause an internal server error when updating the value for the email. When a percent encoded value is present in the URL parameters, PHP decodes the value to the respective character. When the percent encoded value does not correspond to a standard character, MariaDB is unable to perform any comparisons using the value, causing an `Illegal mix of collations` error. Examples of problematic encodings include (but many were found) `%de`, `%bc`, `%cd`.

Mitigation

These problems can be mitigated by performing server side validation of all user input.

6.2.5 Utility Script File Permissions

Summary

The Open XDMoD utility scripts (`xdmod-shredder`, `xdmod-ingestor`, and `xdmod-slurm-helper`) have file permissions that allow any user ID on the metrics host to execute them. However, the scripts will fail when the user does not have adequate permissions to read the necessary configuration file.

Description

All of the utility scripts have permissions `0755/-rwxr-xr-x` that allow any user on the system to execute the scripts. However, the scripts will fail if the user executing them is not in the `xdmod` user group as they will be unable to read the `/etc/xdmod/portal_settings.ini` configuration file.

Mitigation

Though there were no apparent security issues found with this approach, it is recommended that the operating system's privilege restrictions be used in addition by changing the permission bits to `0750/-rwxr-x---`.

6.3 Places searched with no apparent issues found

We evaluated several other components of the system and did not find any problems. Though it is impossible to certify that a code is free of vulnerabilities, we have increased confidence in the security of these parts of the code.

6.3.1 Server Side Validations

Summary

Requests made to the Open XDMoD REST APIs are validated using the cookies sent along with the requests to determine the user the request was made by. We looked for several issues including breach of the trust boundary between the server and client, improper authorization for operations, and bugs in cookie validation.

Location

Files: `html/controllers/*`
`html/internal_dashboard/controllers/*`
`classes/Rest/Controllers/*`

Description

The session cookies sent along with each request identify the user making the request. Once the user has been identified, the user's role and access control list is retrieved from the moddb database. The role and the ACL are used to determine whether the user is authorized to perform the requested operation.

We checked that operations capable of altering data or revealing protected data performed the authorization checks correctly and did not allow unauthorized users to perform the operations. Due to the setup of the custom HTTP controllers in the `html` directory, we also checked that in every case the authorization checks could not be bypassed by making requests directly to the PHP files that perform each operation.

Result

We did not find any apparent issues with the authorization checks that are performed for the privileged operations.

We confirmed that no operation could be performed by simply bypassing the intended PHP file and requesting the PHP file for the operation directly. This is due to the PHP files for each operation not having the necessary include statements in the files themselves.

6.3.2 User Input Sanitization

Summary

Open XDMoD stores user input (including profile information such as name, email, and user name) in its databases for use elsewhere in the portal. We verified that all such user supplied data was properly sanitized before being displayed in the Open XDMoD portal to prevent cross-site scripting vulnerabilities.

Description

We focused on the three ways in which user supplied data can be input into the system:

1. Through the text fields in the Open XDMoD portal including the contact page, signup requests, user profile information, and text fields included in the construction of reports.
2. Directly through the rest API. By bypassing client side validations and restrictions on the content of request parameters ,we tested how Open XDMoD sanitizes the content of request fields that are otherwise a known (or an element of a known set of) value(s).
3. Through the log files processed by either the `xdmod-shredder` or the `xdmod-slurm-helper` script. All the fields were checked for safe handling, especially the user name and job name fields due to their widespread use in the web portal.

Result

We inspected the code handling both the input and the use of the user supplied data. We found that by the time the user supplied data was used in the XDMoD portal, it had been safely escaped, preventing any XSS attacks. In most cases, the escaping was done only at the point that the data was about to be used and was done by the `Ext.util.Format.htmlEncode` function. We note that the escaping was done by using a library supplied function, rather than attempting to sanitize the input with custom code. Using a library is often a more robust solution as libraries are likely to be better tested, more widely used, and cover more corner cases.

6.2.3 SQL Injections

Summary

Open XDMoD utilizes SQL queries to retrieve, insert, and modify data in the data warehouse, to modify user information, and to record log messages. We did an in depth exploration of the construction process of queries made to the databases to ensure that they are not vulnerable to an injection attack. We did not find any unsafe handling of user input that could lead to an injection attack.

Description

SQL queries are constructed dynamically throughout Open XDMoD and are used in both handling of requests from the REST API and the processing of data through the ETL pipeline. Most of the queries that are constructed in response to a request from the REST API use prepared statements for all user provided content, and in doing so eliminate the ability for an injection attack to occur through those fields. However, many of the queries that occur during the ETL pipeline, and a few that occur due to the REST API, use string interpolation in the construction of queries for the purpose of parameterizing the table name(s). Since this cannot be alleviated by using prepared statements, we traced the origin of each variable used to construct the query to verify that it cannot be tainted by user input.

Result

In every case where string interpolation was used to construct a query we verified that the value of the variable could not be controlled by the user. In every case, we were able to trace the origin to either string literals defined higher up the stack, or to the JSON files that define the ETL pipeline. Since these files are only writable by root, they are considered a trusted source.

Appendices

Appendix A: OpenXDMoD-2020-0001

Appendix B: OpenXDMoD-2020-0002

Appendix C: OpenXDMoD-2020-0003

Appendix D: OpenXDMoD-2020-Warning-0001

Appendix A

Distribution: *Release to XDMoD developers only*
Release to public after Dec 1 2020

OpenXDMoD-2020-0001

Summary:

The default configuration of Open XDMoD does not encrypt HTTP traffic. This allows attackers to monitor all traffic between the server and the client. As a result, passwords submitted on login are sent in plain text, and the `xdmod_token` cookie can be stolen from any HTTP request after authentication.

Component	Vulnerable Versions	Platform	Availability	Fix Available
Open XDMoD	All	All	n/a	n/a
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	Network	Any	Medium	High
Fixed Date	**Credit**			
n/a	Ben Kinzer Ian Ruh			

Access Required: Network

An attacker needs no special permissions to observe the unencrypted traffic, only access to a packet sniffing tool (e.g. Wireshark [1]) and a connection to the client or server network.

Effort Required: Medium

To observe the traffic using packet sniffing tools like Wireshark, an attacker must have access to the network over which the traffic is sent. There are many ways to access an insecure network including physically tapping network wires/devices, connecting to the same insecure wireless network as a client, and more. The `xdmod_token` is sent in plain text with every authenticated request. On login, user credentials are submitted as plain text form entries.

Impact/Consequences: High

The impact of exploiting this vulnerability depends on the victim whose traffic is being observed. The attacker will have the login credentials of the victim and thus have the ability to login as the victim and perform any action on the victim's behalf. In the case of a victim with a manager role, the attacker could create arbitrary accounts and view XDMoD logs.

Full Details:

1. Session Hijacking

An attacker can observe the `xdmod_token` sent with a legitimate client's request. While `xdmod_token` is still valid, the attacker can append this cookie to their own malicious requests and thus perform any action the victim is authorized to perform. Figure 1 shows the `xdmod_token` value that is readable from the HTTP request header. This figure is a screen capture from Wireshark.

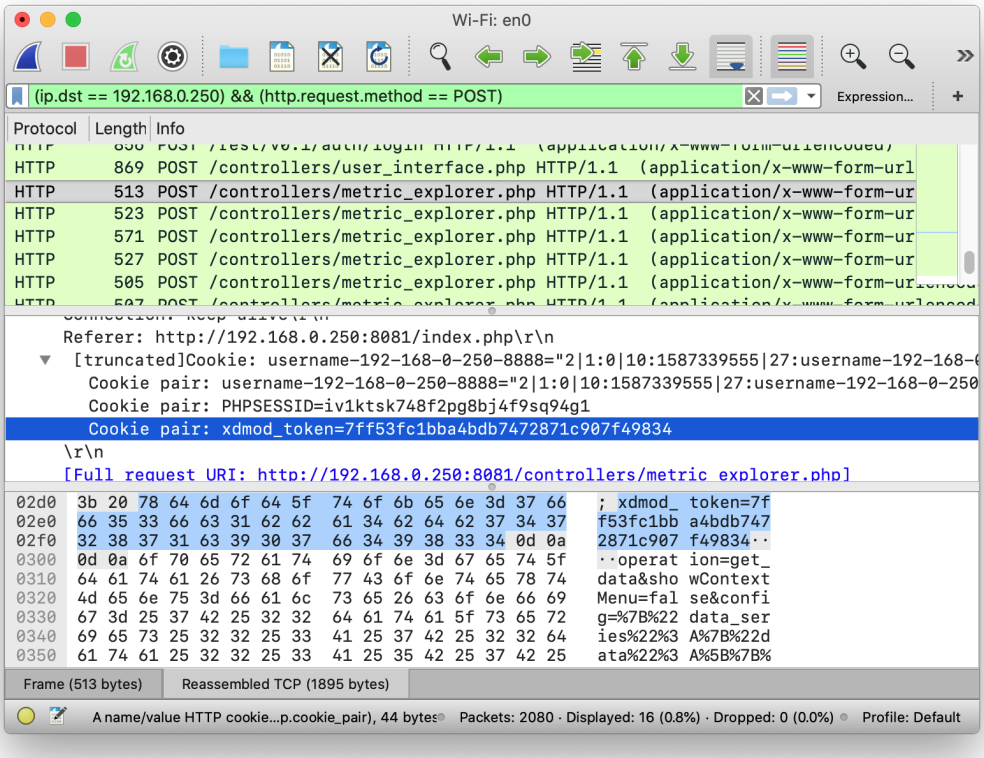

Figure 1. `xdmod_token` visible in the headers of an unencrypted HTTP request.

2. Password Sniffing

If an attack observes a login form POST request, they will be able to see the plain text username and password. The attacker can then freely login as the victim and perform operations on their behalf. Figure 2 shows the plain text username and password after being sniffed by Wireshark.

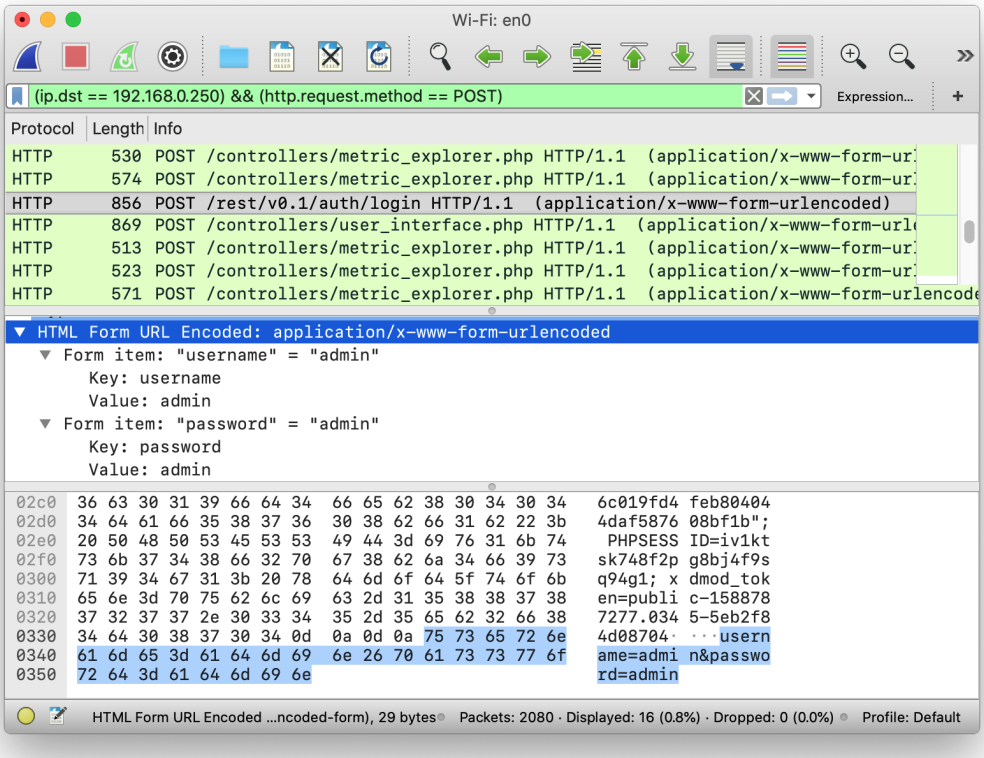

Figure 2. Plain text username and password visible in unencrypted POST request.

3. Behavior Monitoring

   All of the operations performed by a legitimate user over HTTP can be observed by an attacker, therefore revealing possibly sensitive information without ever interacting with the system.

Cause: Unconventional Encryption Design

In both the session hijacking and password sniffing vulnerabilities HTTPS is not enforced for all server traffic.

Proposed Fix:

All three of the possible exploits above can be mitigated by enforcing HTTPS on all traffic to/from the server. This could be achieved by configuring Apache by default to only allow HTTPS and by providing a self-signed certificate until the user replaces it with their own. Also ensure that all cookies are sent with the secure flag. This flag prevents cookies from being sent over unencrypted channels. More details about the secure flag are described in [2].

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.

References:

[1] Wireshark. Wireshark Foundation. https://www.wireshark.org/
[2] "Secure Cookie Flag". OWASP Foundation. Web. Accessed 6 May 2020. https://owasp.org/www-community/controls/SecureFlag

Distribution: *Release to XDMoD developers only*
Release to public after Dec 1 2020

Appendix B

Distribution: *Release to XDMoD developers only*
Release to public after Dec 1 2020

OpenXDMoD-2020-0002

Summary:

If an attacker has a normal user account on the metrics server host, they would be able to hijack any currently active session. Every time a request is made to the XDMoD API, the metrics server attempts to verify the `xdmod_token` cookie belongs to a valid session. In querying the `moddb` database, the query, and the parameters of the query (including the cookies being checked), are logged to the `/var/log/xdmod/session_manager.log` file. This log file is globally readable. By reading the log file an attacker can see any active session cookies and therefore be able to impersonate any of the active sessions.

Component	Vulnerable Versions	Platform	Availability	Fix Available
Open XDMoD	All	All	n/a	n/a
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	Account on Host	Any	Low	Low
Fixed Date	**Credit**			
n/a	Ian Ruh			

Access Required: Account on Host

An attacker needs an account on the host running the metrics server or access to the metrics server's filesystem to be able to read the `session_manager.log` file, and an HTTP client to hijack an active session.

As the documentation for XDMoD [1] states that "For security reasons it is suggested, but not required, that no other services run on the same machine and that access is limited to system administrators," if the system is setup as suggested by the documentation, this vulnerability would not be easily exploitable.

However, even if the metrics server were running on a dedicated host, if it were using a shared file system to store log files then the files would be accessible to anyone else able to access the shared

filesystem.

Effort Required: Low

Assuming an account on the metrics server host or access to the metrics server's filesystem, an attacker can read the `session_manager.log` file to get the cookies of all active sessions. They can then hijack any active session by setting the stolen session cookies on an HTTP client of their own and using that client to impersonate a legitamate user.

Impact/Consequences: Low

The impact of exploiting this vulnerability depends on the victim whose session is hijacked. Once the session has been hijacked, the attacker can perform any action on the victim's behalf. In the case of a victim with a manager role, the attacker could create arbitrary accounts and view XDMoD logs.

This impact was labelled "low" because the attack will come from a user who already has an account on the system.

Full Details:

Once an attacker has access to the any account on the machine, they are able to read the globally readable `/var/log/xdmod/session_manager.log` file to get the cookies for all active sessions. By setting those cookies in their HTTP client, the attacker can hijack any active session.

Additionally, while critical information is not being logged to them, several other log files are also globally readable, possibly exposing unintended information to other users of the system. These log files (all in `/var/log/xdmod/`) are `apache-access.log`,`apache-error.log`, `exceptions.log`, and `query.log`.

Figure 1 shows the `session_manager.log` file being read by a non-root, non-sudoer user. Figure 2 shows the cookies retrieved from the log file being set in the Firefox dev tools. Figure 3 shows the successfully hijacked session after the page was reloaded.

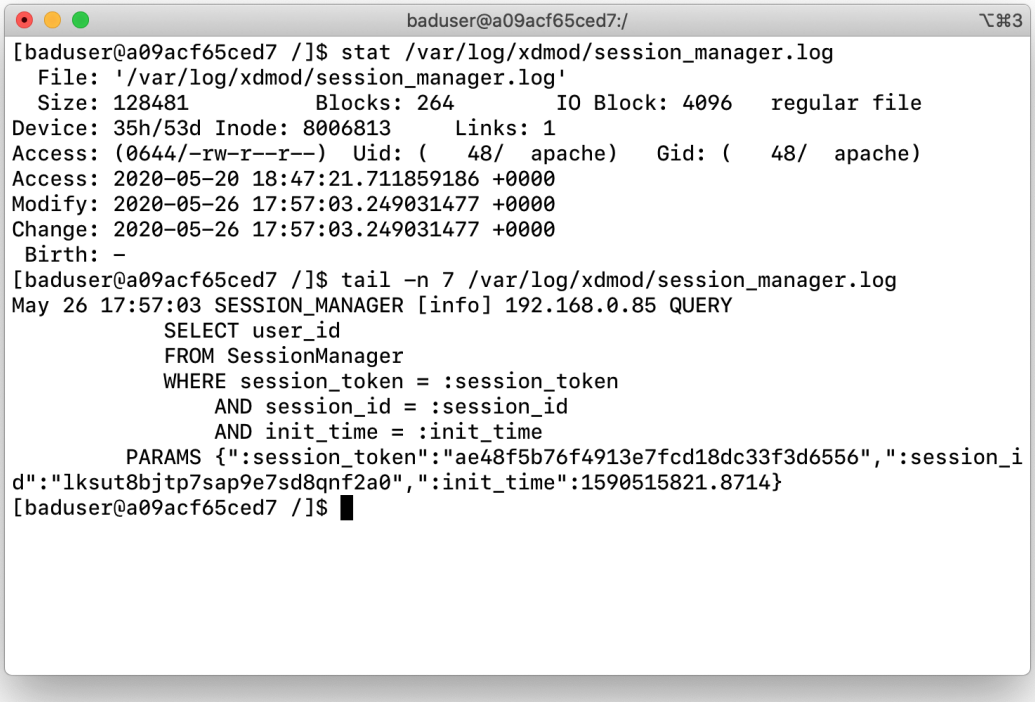

Figure 1. The `session_manager.log` file being read by a non-root, non-sudoer user. In the file, the `session_token` field corresponds to the `xdmod_token` cookie, and the `session_id` corresponds to the `PHPSESSID` cookie.

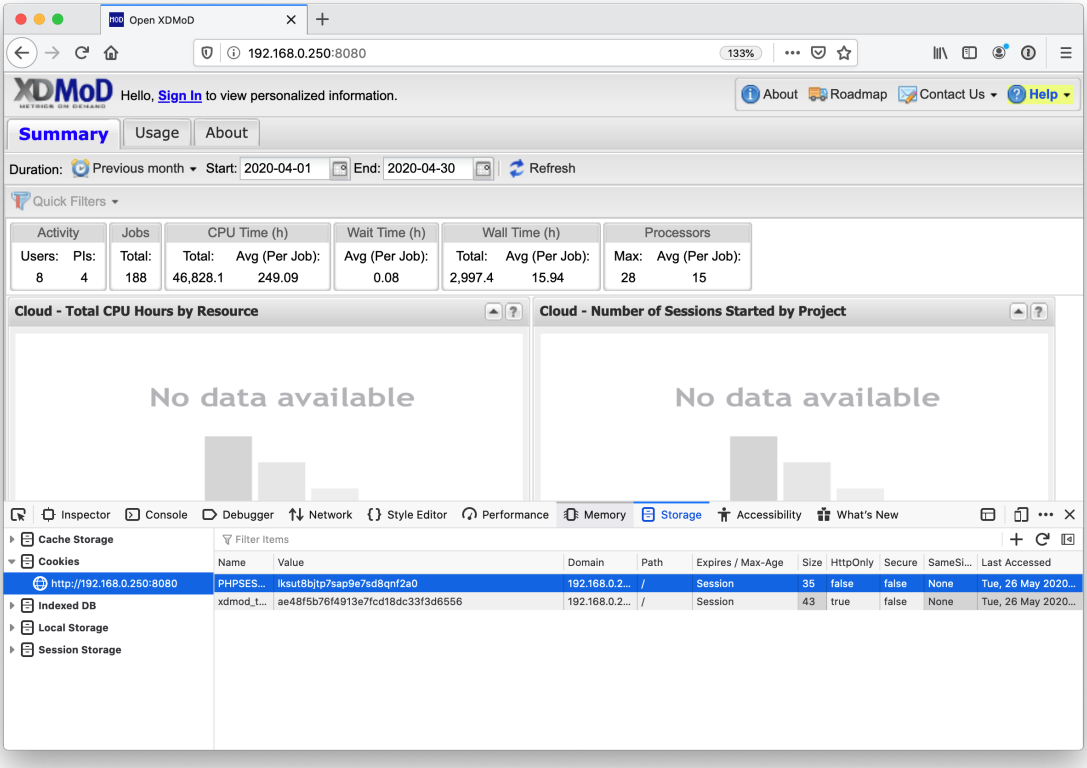

Figure 2. The cookies retrieved from the log file being set in the Firefox dev tools.

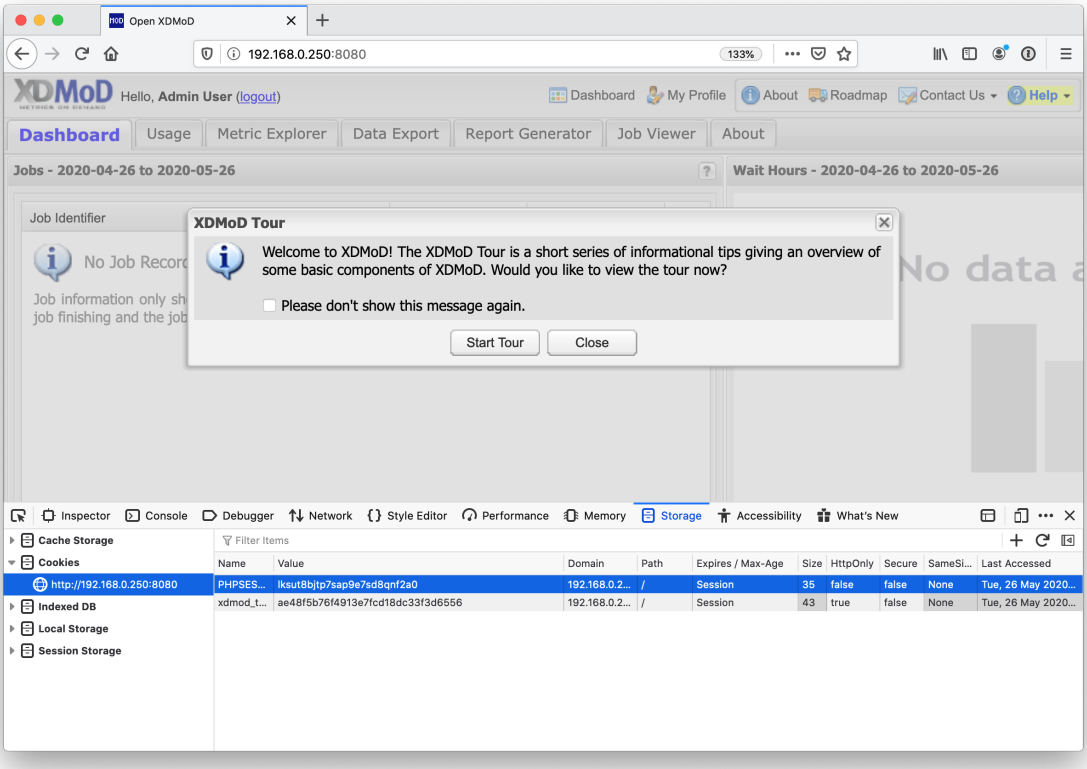

Figure 3. The successfully hijacked session after the page was reloaded.

The exploit demonstration shown above was carried out by initially logging in as the "Admin User" on one machine to create an active session that could be hijacked. From a seperate machine, the `session_manager.log` file was accessed by ssh'ing into the metrics server host. On the second machine, the cookies retrieved from the log files were set as the cookies in Firefox (Figure 2). When the page for the portal was reloaded (Figure 3), the second machine now had access to the "Admin User" account used to create the session on the first machine.

Cause: Unsafe File Permissions

This vulnerability was caused by the unsafe file permissions set on the log file.

Proposed Fix:

The vulnerability can be easily fixed by changing the file permissions from `0644/-rw-r--r--` to `0640/-rw-r-----`.

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.

References:

[1] Open XDMoD 8.5 Hardware Requirements. UBCCR. https://open.xdmod.org/8.5/hardware-requirements.html

Distribution: *Release to XDMoD developers only*
Release to public after Dec 1 2020

Appendix C

Distribution: *Release to XDMoD developers only*
Release to public after Dec 1 2020

OpenXDMoD-2020-0003

Summary:

XDMoD is vulnerable to a Denial of Service (DoS) attack that prevents proper functioning of the XDMoD portal. Every time a request is made to Apache, an entry is logged to `/var/log/xdmod/apache-access.log` containing the timestamp, HTTP method, route, and client information. By repeatedly sending requests (on any route) to Apache, an attacker can fill up the free space on the file system partition. This results in the XDMoD web portal failing to properly load due to the malformed page returned by the server. In addition to XDMoD, any other programs that use the same file system partition as the log files may also be affected.

Component	Vulnerable Versions	Platform	Availability	Fix Available
Open XDMoD	All	All	n/a	n/a
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	Network	Any	Low	Medium
Fixed Date	**Credit**			
n/a	Ian Ruh			

Access Required: Network

An attacker must be able to address the host that runs the metrics server's web portal.

Effort Required: Low

An attacker needs a scriptable HTTP client. The attacker can then flood the server with requests until the web portal fails to properly load. The length of time this requires depends on the speed at which the attacker can make requests, the size of the metrics server's file system partition, and the route(s) that the attacker requests.

Impact/Consequences: Medium

The impact of exploiting this vulnerability is rendering the XDMoD web portal inoperable and hindering any users of the file system partition on which the log files reside until disk space can be freed. However, as the Hardware Requirements section [1] of the documentation recommends "that no other services run on the same machine," if the recommendation is followed, then the failure of other services on the system is secondary to the failure of XDMoD.

If `/var/log` resides on a shared network file partition, then there could be an impact on the functionality of other programs running on other hosts that share the same network file system partition.

Full Details:

An attacker with access to the network can use an HTTP client, such as the command line utility bombardier [2], to repeatedly make requests to the Apache server. Every time a request is made, an entry is logged to `/var/log/xdmod/apache-access.log` containing the timestamp, HTTP method, route, and client information. With a sufficient number of requests, this file can be made to grow until it uses all free space available on that file system partition. After all free space has been used, though HTTP requests still succeed, the PHP program generating the response fails to return a properly formatted HTML document, resulting in a blank page. From the raw return value of the requests, it is evident that MariaDB is failing with the following error: `SQLSTATE[HY000]: General error: 1 Can't create/write to file '/var/tmp/#sql_4db_4.MAI' (Errcode: 28)`, which corresponds to 'No space left on device'.

Using a VM with 6 GB of free space after the installation of XDMoD, we verified this attack by using the command `bombardier -c 20 -n 1000000 http://[host]:[port]/[path]` from a separate machine. This command sets up 20 simultaneous connections and makes 1,000,000 requests to the specified URL. In our testing, we used a trick to increase the speed at which the file system partition is filled by requesting a path that consists of 100's of random letters. This had two effects: 1.) Since the requested path is logged to `apache-access.log` on every request, we are able to write more data for a given request, and 2.) If the path is longer than 255 characters, then Apache will write the entire path and the error message that the file name is too long to the `/var/log/xdmod/apache-error.log` file, therefore nearly doubling the rate at which the file system partition is filled. On a local gigabit network, we were able to write ~78 MB/s to disk, filling up the remaining free space in less than three minutes. It may be possible to increase the throughput by using different options or a different HTTP client, so 78 MB/s should be considered a lower bound.

XDMoD's current method for handling the rotation of log files using `logrotate` does not prevent this attack as the time span in which this attack is possible (minutes to hours) is far less than the period at which `logrotate` is configured to rotate logs (weekly).

Cause: Unchecked Log File Size

This vulnerability was caused by the unlimited growth of log files.

Proposed Fix:

One option to fix this vulnerability is to use the Apache feature of piped logs and the Apache provided utility `rotatelogs` [3], which allows continuous monitoring and control of file size.

An additional mitigation is to isolate log files to their own file system partition. In the case that these

files grow to a large size, this will isolate the effect and reduce the impact. This approach can be used with any other approach to add an additional layer of protection.

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.

References:

[1] Hardware Requirements. Open XDMoD. https://open.xdmod.org/8.5/hardware-requirements.html
[2] bombardier. codesenberg/bombardier. https://github.com/codesenberg/bombardier
[3] Piped Logs. Apache HTTP Server Project. http://httpd.apache.org/docs/2.4/logs.html#piped

Appendix D

Distribution: *Release to XDMoD developers only*
Release to public after Dec 1 2020

OpenXDMoD-2020-Warning-0001

Summary:

The current version of XDMoD has three problematic dependencies. Two of the dependencies are unmaintained and the third is outdated and vulnerable (though it is not clear if it is exploitable in XDMoD). XDMoD uses the no-longer-maintained HTTP framework Silex that had an end-of-life in June of 2018 [1]. Although there are currently no publicly known vulnerabilities in Silex, the lack of active development and maintenance increases the likelihood that vulnerabilities will not be caught and will not be fixed. XDMoD also relies on PhantomJS, development of which has been suspended [2], and which has a publicly known vulnerability. Additionally, XDMoD uses an outdated and vulnerable version of the JasperReports Library Community Edition (where the vulnerability reports are currently too terse to tell if they affect XDMoD).

Component	Vulnerable Versions	Platform	Availability	Fix Available
XDMoD	n/a	All	Public	n/a
Status	**Access Required**	**Host Type Required**	**Effort Required**	**Impact/Consequences**
Verified	n/a	n/a	n/a	n/a
Fixed Date	**Credit**			
n/a	Ian Ruh			

Access Required: n/a

Effort Required: n/a

Impact/Consequences: n/a

The CVE for PhantomJS is not exploitable because the use of PhantomJS in XDMoD is limited. The only JS file executed by PhantomJS is `libraries/phantomjs/generate_highchart.js`, so an attacker would need to modify the file to exploit the vulnerability.

As there are no publicly available details about the CVEs affecting JasperReports, we have not been able to conclude anything about the impact or consequences for XDMoD.

Full Details:

Silex

The current version of XDMoD uses the HTTP framework Silex, the end-of-life of which passed in June 2018 and has not been updated since. Additionally, the version in use by XDMoD is version 1.3.6, while the most recent version is 2.3.0. However, there are no publicly known vulnerabilities at this time.

PhantomJS

XDMoD relies on PhantomJS as part of generating periodic reports. The development of PhantomJS has been suspended since March 2018 and there is one publicly available vulnerability for the current version.

- CVE-2019-17221: Arbitrary File Read Vulnerability
  An attacker can craft JS that, when executed, can read arbitrary files on the system. Due to the limited use of PhantomJS in XDMoD, this vulnerability is not exploitable unless an attacker already has root privileges.

JasperReports Library Community Edition

XDMoD relies on an outdated and vulnerable version of the JasperReports Library CE as part of generating periodic reports. The version of JasperReports being used is v3.7.6, while the current version is v6.12.12 [3]. v3.7.6 suffers from the following publicly disclosed vulnerabilities:

- CVE-2018-18809: Directory Traversal Vulnerability
- CVE-2018-5429: Arbitrary Code Execution Vulnerability
- CVE-2017-5529: Information Disclosure Vulnerability

Due to the lack of publicly available details about the CVEs above, we cannot determine whether they are exploitable in XDMoD, and, if so, the impact or consequences.

Cause: Unmaintained Software Component

One of the most common security flaws in any software system is the use of components with known vulnerabilities [4]. As vulnerabilities are discovered and patched in software components, it is the responsibility of software developers who use those components to ensure that their software includes the most up-to-date components. In the case when software is no longer being maintained, even critical vulnerabilities may not be fixed. This exposes dependent software to an increased risk of attack.

Proposed Fix:

Silex

Using actively maintained software lowers the risk of vulnerabilities going uncaught and unfixed.

The recommendation from the developers of Silex is to migrate to using Symfony and Flex [5], which continue to be actively maintained and developed.

PhantomJS

Using actively maintained software ensures that publicly known vulnerabilities are fixed and do not endanger dependent software to being exploited. It is recommended that an actively maintained alternative be found with equivalent functionality to replace PhantomJS.

JasperReports Library Community Edition

Keeping all libraries and frameworks updated to their most recent versions prevents numerous vulnerabilities. Jaspersoft has fixed these vulnerabilities in newer versions of JasperReports Library CE. At the time of writing this report, JasperReports Library CE v6.12.12 is secure from the vulnerabilities described above.

Acknowledgment:

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

References:

[1] Silex. silexphp/Silex. https://github.com/silexphp/Silex
[2] PhantomJS. ariya/phantomjs:#15344 https://github.com/ariya/phantomjs/issues/15344
[3] JasperReports Library. Jaspersoft Community. https://community.jaspersoft.com/project/jasperreports-library
[4] "Top 10 2019-A9-Using Components With Known Vulnerabilities". OWASP. Web. Accessed 11 May 2020. https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities
[5] "The end of Silex". Symfony. Web. Accessed 11 May 2020. https://symfony.com/blog/the-end-of-silex