Hardening Web Applications Using a Least Privilege DBMS Access Model

Stuart Steiner
Ctr. Secure & Dependable Sys.
University of Idaho
Moscow, Idaho
ssteiner@uidaho.edu

Daniel Conte de Leon
Ctr. Secure & Dependable Sys.
University of Idaho
Moscow, Idaho
dcontedeleon@ieee.org

Ananth A. Jillepalli
Ctr. Secure & Dependable Sys.
University of Idaho
Moscow, Idaho
ajillepalli@ieee.org

ABSTRACT

Within the last three years hundreds of millions of private data records have been compromised in high-profile data breaches, resulting in billions of dollars in economic losses and unrecoverable loss of privacy. One commonality is that attackers obtained administrative-level access to records on a central database. We argue that the widespread practice of highest privilege design and configuration is a significant contributor, where users and applications are given the highest level of privilege needed to execute the union of all needed tasks. One problematic common practice is, in a web-based application, for front-end and middleware processes to have root privileges to the complete DBMS back-end database. This practice is in stark opposition to the well-known secure design principle of least privilege introduced 40 years ago. Enforcing least privilege at all levels of a web application would help prevent future all-lost cyber-compromises. Here we introduce Hierarchical Policy (HPol), a formal access control modeling tool used in modeling web application database security.

CCS CONCEPTS

• **Security and privacy → Information flow control**; *Vulnerability scanners*;

KEYWORDS

security, database security, web application security

ACM Reference Format:
Stuart Steiner, Daniel Conte de Leon, and Ananth A. Jillepalli. 2018. Hardening Web Applications Using a Least Privilege DBMS Access Model. In *CyberSec '18: Fifth Cybersecurity Symposium, April 9–11, 2018, Coeur d' Alene, ID, USA*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3212687.3212863

1 INTRODUCTION

Using the web is now part of everyday life. Unfortunately, another part of everyday life is the high likeliness of web accessible private data to be stolen or lost. Since the start of 2017, over 900 companies in the United States of America and over 1100 companies world wide have experienced data breaches[12]. Billions of private records have been compromised and it is estimated that billions of dollars have been lost. One of the emerging common patterns in these breaches is that attackers gained complete access to the database back-end by exploiting vulnerabilities in the web application or the middleware.

1.1 Problem and Approach

A major root problem contributing to many of these breaches, is the widespread usage of the highest privilege design pattern. In such a pattern, users and applications are given the highest level of privileges needed to execute the union of all needed tasks. In the case of web applications, this design pattern translates into the practice of granting the web application and/or middleware processes root privileges over the back-end database management system (DBMS). The same pattern can also used to grant the middleware processes root-level privileges over the file system within the web application server.

Because of the problem described above, once a web application has been compromised an attacker can easily gain root-level access to the back-end database. Using such design pattern violates two of the most basic principles of secure system design: (1) Least Privilege and (2) Layering or Defense in Depth.

The hardening approach described in this paper specifically targets locations where a highest privilege design pattern has been used. In this paper, we describe our approach using a case study based-on the Open Web Application Security Project (OWASP) Mutillidae [8, 19] web application. We analyzed the Mutillidae source code, built a non-least privilege model, converted the model to least privilege, and then modified the Mutillidae source code to implement a least privilege design pattern. The result is two formal security models: a current and a least privilege model, and a more secure web application.

1.2 Contribution and Overview

The contribution of this article is a step-by-step and formal process for applying the principle of least privilege design pattern to a legacy web application written in PHP with embedded SQL.

The remainder of this article is organized as follows: In Section 2 *Background*, we analyze the current state of web application security and remind readers about the well-known principle of least privilege; In Section 3 *Building a Web Application Security Model*, we describe how to construct a security model for a web application using the hierarchical security modeling framework (HPol); In Section 4, *Applying Least Privilege to Model and Application*, we describe how to convert the previous model from a highest privilege model

CyberSec '18, April 9–11, 2018, Coeur d' Alene, ID, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6406-5/18/04.
https://doi.org/10.1145/3212687.3212863

into a least privilege and more secure model. We also describe how to convert the application code to match the least privilege model. In Section 5 *Related Work*, we present related work. In Section 6 *Conclusion and Future Work*, we summarize the problem, approach, and results, and list avenues of ongoing and future work.

2 BACKGROUND

This section describes the current state of web application security, the secure design principle of least privilege, and the hierarchical security policy formal model called HPol.

2.1 Current State of Web Application Security

It is agreed by the cybersecurity community that most web applications in use today are not designed, developed, and maintained following secure design best-practices. There may be many reasons for how we have arrived at such insecure state of the web. Below, we present three reasons that we believe have led to the current lack of security and possible avenues for improvement.

(1) **Rapid Growth and Lack of Security Best Practices:** The Web has evolved over the years and also grown very rapidly. Currently, the web is driven by about two billion websites [17] and many millions of lines of shared and legacy code. Many, if not most, of these lines of code were crafted without following adequate secure design principles. It would be impossible to manually and promptly secure all this code for the following two reasons: (1) Lack of availability of trained personnel and (2) Extensive time and financial resources needed for manual hardening or code replacement.

(2) **Lack of Adequately Trained Personnel:** Currently, a majority developers of web applications have less than five years of experience (51%) [1, 18] and lack a holistic understanding of the complete system driving most web applications. Sinclair and Smith state that *Effective security requires looking at an entire system* [23]. In addition, currently, most web developers are self-taught (61%) with reduced expertise or experience in secure design, development, and implementation techniques [4, 18].

(3) **Not Following Secure Design Principles:** Perhaps due to the two reasons stated above, and many others, today, most web applications are not designed or implemented following Saltzer and Schroeder's secure design principles introduced decades ago [21]. One principle that would enable the current web to better mitigate current attacks is the *Principle Of Least Privilege* (POLP). Helping implement the POLP for the current Web is the focus of the research reported in this paper.

Using PHP as an example, a PHP database class is usually built to acquire a connection to the DBMS. This class typically uses a single user to connect to the database back-end and that user is usually *root* on a DBMS such as MySQL or MariaDB. Figure 1 illustrates example PHP code to allow the web application process to login into the MySQL database. This code excerpt appears in a popular web development tutorial [20]. More than 70% of active websites use PHP [25].

INSECURE HIGHEST PRIVILEGE PERMISSIONS			
Webpage Name (Subject)	**DBMS User (Subject)**	**Granted Permission (Action)**	**DBMS DB Table (Object)**
Non-least privilege permissions:			
ALL pages	root	ALL (*.*)	ALL.ALL

Table 1: The most common access control design pattern is to grant the middleware *root* access to the complete DBMS. This grants all permissions on all objects on all databases in the DBMS.

The majority of online code tutorials show code implementing the same pattern which implements a highest privilege approach to access control. Under this highest privilege design pattern, if any page within the web application has a vulnerability and code can be maliciously injected or remotely executed, then, the attacker will be able to run the code of their choice with root privileges.

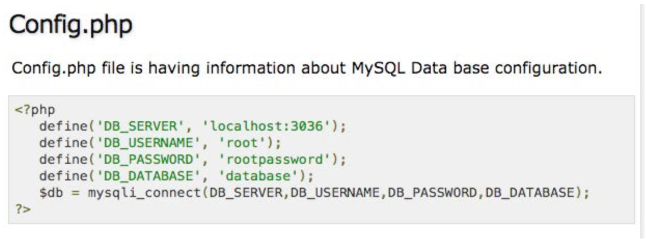
2.2 The Principle of Least Privilege

The principle of least privilege (POLP) is a secure systems design principle originally described by Saltzer and Schroeder in 1975 [21]. The POLP states that each system component (people or process) should be given the least possible authority that is necessary to perform the task at hand. Implementing such principle would help reduce vulnerabilities and also mitigate damages from exploits and compromises [2, 9, 22, 26].

In the case of a web application, for example, an `index.php` web page that only displays fixed information, and no information from the back-end database, must not be granted access privileges to the back-end database server (DBMS). If the POLP was applied, then, a successful attack on the `index.php` page will not automatically result in a complete compromise of the DBMS. Similarly, if a `showBlog.php` web page needs `SELECT` permission for the *blogs* table within the *mutillidae* DBMS database, then the middleware process executing the `showBlog.php` code, must only receive only `SELECT` level access, and no more.

2.3 The HPol Formal Model

The Hierarchical Policy (HPol) formal model enables the representation of access control and security policies using a hierarchical


Config.php

Config.php file is having information about MySQL Data base configuration.

```
<?php
   define('DB_SERVER', 'localhost:3036');
   define('DB_USERNAME', 'root');
   define('DB_PASSWORD', 'rootpassword');
   define('DB_DATABASE', 'database');
   $db = mysqli_connect(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);
?>
```

Figure 1: Online code tutorial illustrating setting root as the single user to the database (Code excerpt from [20], GNU License)

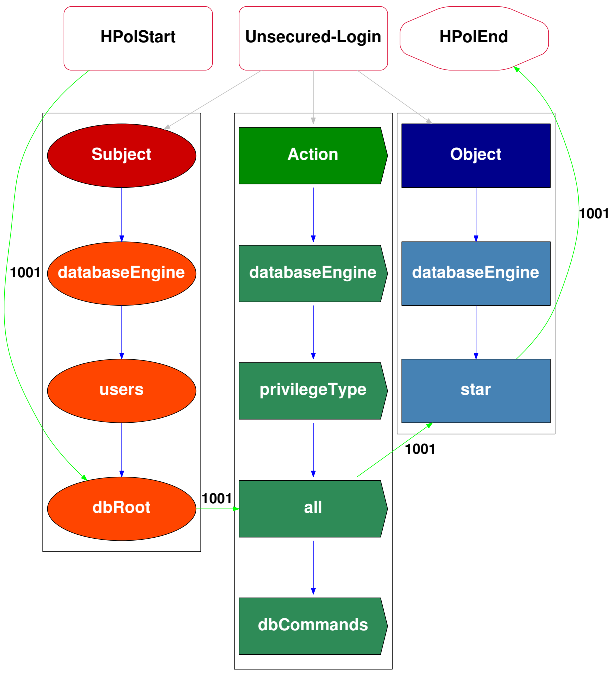
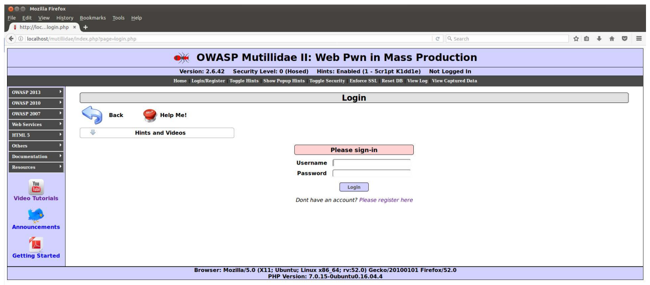


Figure 2: HPol Example illustrating an non-least privilege database user dbRoot has full access to the entire database system.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://loc...login.php
localhost/mutillidae/index.php?page=login.php
Search
Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources
Video Tutorials
Announcements
Getting Started
Login
Back
Help Me!
Hints and Videos
Username
Password
Login

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.42 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Please sign-in

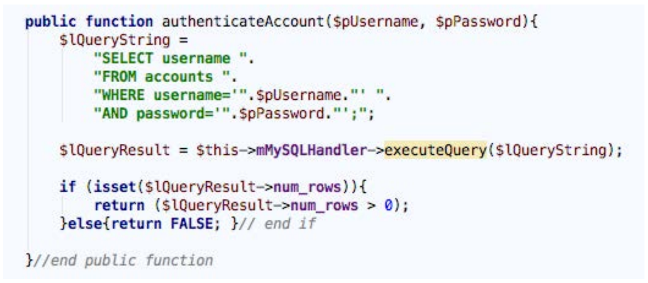
Dont have an account? Please register here

Browser: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
PHP Version: 7.0.15-0ubuntu0.16.04.4

Figure 3: The Mutillidae login.php page (Code excerpt from [8], GNU License).

graph structure. In HPol subjects, actions, and objects are represented by a hierarchical graph. Policies are represented by links, or relation tuples, that state that a given subject has been granted permission to perform a given action on a given subject. HPol represents each subject, action, and object with a node within a directed acyclic graph (DAG). Policy links between nodes connect subject, action, object nodes within the DAG to indicate what policies are allowed [6].

Figure 2 shows a portion of the resulting HPol model for the highest privilege design of the Mutillidate web application. In the model the user dbRoot is granted administrative-level permissions for all objects: tables, functions, and procedures within all databases within the DBMS. Policy number 1001 indicates the granting of such permissions to the corresponding subject, action, object tuple.



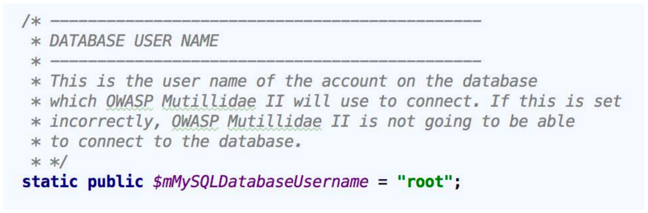
```
public function authenticateAccount($pUsername, $pPassword){
    $lQueryString =
        "SELECT username ".
        "FROM accounts ".
        "WHERE username='".$pUsername."' ".
        "AND password='".$pPassword."';";

    $lQueryResult = $this->mMySQLHandler->executeQuery($lQueryString);

    if (isset($lQueryResult->num_rows)){
        return ($lQueryResult->num_rows > 0);
    }else{return FALSE; }// end if

}//end public function
```

Figure 4: The database query generated from login.php that allows the user to login (Code excerpt from [8], GNU License).



```
/* ------------------------------------------------
 * DATABASE USER NAME
 * ------------------------------------------------
 * This is the user name of the account on the database
 * which OWASP Mutillidae II will use to connect. If this is set
 * incorrectly, OWASP Mutillidae II is not going to be able
 * to connect to the database.
 * */
static public $mMySQLDatabaseUsername = "root";
```

Figure 5: PHP Code illustrating a single privileged root database user (Code excerpt from [8], GNU License).

3 BUILDING A WEB APPLICATION SECURITY MODEL

In this section the HPol security model is applied to the database access portion of the Mutillidae web application. To create this security model the Mutillidae web application source code was analyzed and the subjects, actions, and objects were identified. In this instance, the subjects are the individual web pages and the Mutillidae database user (dbRoot). The actions are identified as the MySQL database command allowed on the Mutillidae database. The objects are identified as the individual web pages of the Mutillidae web application and the Mutillidae MySQL database tables.

3.1 HPol Mutillidae Database Access Model

For the Mutillidae web application the database management system (DBMS) is *MySQL*. The set of available MySQL commands are, `show databases`, `select`, `alter`, `drop`, `create`, `delete`, `insert`, `shutdown`, `process`, `execute`, and `update`. The Mutillidae web application contains a file *classes/MySQLHandler.php* with a single privileged user named root. This root user has full access to all commands in the DBMS. Figure 5 illustrates the creation of the single privileged root user in the file *MySQLHandler.php*.

Figure 2 illustrates the described MySQL database structure. This figure illustrates that root user can perform any database command on any database, table, function or procedure within the Mutillidae MySQL DBMS. In the HPol Mutillidae security model the dbRoot user is the subject. The actions are the possible SQL queries such as `select`, `update`, `insert`. The objects are a hierarchical structure under the MySQL DBMS. At the top of hierarchy is the regular expression `star` which represents zero or more databases or database tables. Below `star` is `starDotStar`. Below `starDotStar`

HARDENED LEAST PRIVILEGE PERMISSIONS			
Webpage Name (Subject)	**DBMS User (Subject)**	**Granted Permission (Action)**	**Mutillidae DB Table (Object)**
index.php	none	none	none
login.php	selectUser	select	accounts
userInfo.php	selectUser	select	accounts
viewBlog.php	selectUser	select	accounts blogs_table
addToBlog.php	selectInsertUser	select insert	accounts blogs_table
register.php	selectInsertUser	select insert	accounts accounts
captureData.php	insertUser	insert	capture_data

Table 2: In a hardened and least privilege design pattern the access control is restricted to the minimum level of permissions needed for the middleware to perform the required action. For this, new DBMS users with adequate roles must be created and least privilege grant permissions configured in the DBMS.

is the `mysqlDotStar`. Below `mysqlDotStar` are the database *tables*, *functions* and *procedures* for that database.

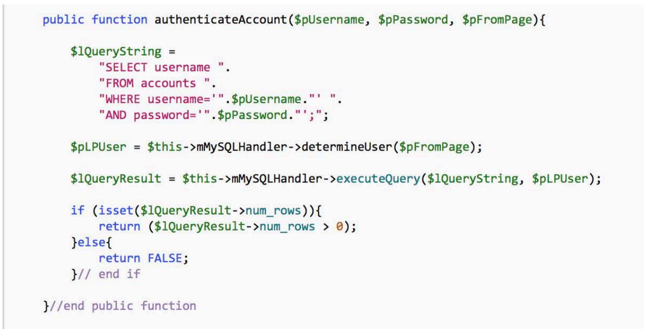
A Mutillidae web page executes a query into the database via the PHP file *classes/SQLQueryHandler.php*. The web page calls *SQLQueryHandler.php* that constructs the MySQL query. Once the query is constructed the HPol Mutillidae subject is a call to the database made by the *MySQLHandler.php* root user. The action is the SQL query. The HPol Mutillidae object is the database table, database procedure, or the database function referenced by the SQL query.

For example an end-user attempting to login to their Mutillidae account would execute the page *login.php*. This login page prompts the user for their username and password. Once the username and password are entered and the submit button is pressed the function *authenticateAccount* is called in the file *SQLQueryHandler.php*. This PHP file builds the SQL query. Figure 3 displays the Mutillidae login screen. Figure 4 illustrates the constructed SQL query after the user input their username and password on the *login.php* page.

4 APPLYING LEAST PRIVILEGE TO MODEL AND APPLICATION

The previous section discussed how to create a formal database security model of the Mutillidae web application using HPol. Now the task is moving the Mutillidae database security model from non-least privileged to least privileged by methodically applying the principle of least privilege.

Step 1: Identify the DBMS and Mutillidae database permissions required to move from non-least privilege to least privilege. Table 1 illustrates the current non-least privilege access control permissions. The appropriate database permissions, including least privilege



```
public function authenticateAccount($pUsername, $pPassword, $pFromPage){

    $lQueryString =
        "SELECT username ".
        "FROM accounts ".
        "WHERE username='".$pUsername."' ".
        "AND password='".$pPassword."';";

    $pLPUser = $this->mMySQLHandler->determineUser($pFromPage);

    $lQueryResult = $this->mMySQLHandler->executeQuery($lQueryString, $pLPUser);

    if (isset($lQueryResult->num_rows)){
        return ($lQueryResult->num_rows > 0);
    }else{
        return FALSE;
    }// end if

}//end public function
```

Figure 6: The database query with POLP applied generated from login.php that allows the user to login. The page making the query is sent to a method which determines the least privilege user. In this case the selectUser (Code excerpt based on modified version of the code from [8], GNU License).

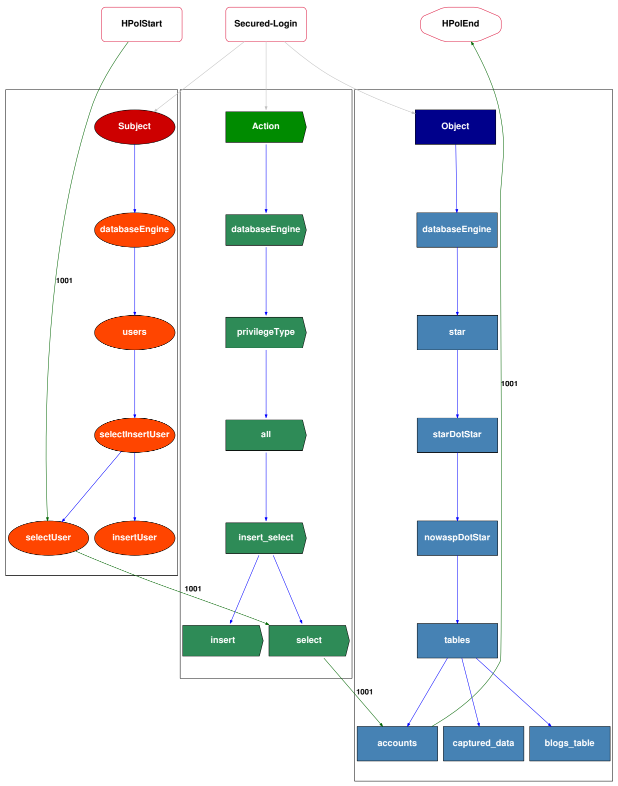


Figure 7: The HPOL Least Privilege model for database access.

database users, and least privilege database commands for each web page in the Mutillidae web application are displayed in Table 2.

For example, the *index.php* page only displays information on the Mutillidae web application, and it contains links to open other web pages. From Table 1 the column labeled `Granted Permissions` shows that the *index.php* page allows access to all database command as well as access to all database tables. The principle of least privilege states there are no appropriate database command or database tables that should be accessed from the *index.php* page. The enforced POLP database access, from Table 2, is shown in the column labeled `Granted Permissions` and the column labeled `Mutillidae DB Table`. Furthermore, the column labeled `Mutillidae DB Table` specifies the tables that are allowed access from the user specified in the column labeled `DBMS User`. For this instance of the web page *index.php* the page does not have access to any database commands or to any database tables.

Step 2: The single privileged root user of the database, shown in Figure 5, is abandoned and new least privilege users are created based on the purpose of the individual web page. The appropriate HPol security model for the POLP requires that a single least privileged user is created. For example the HPol subject `selectInsertUser` is broken into individual users of `selectUser` and `insertUser`.

For example the *login.php* page, from Figure 3, prompts the user to enter their username and password. Recall that once the username and password are entered and the submit button is pressed, the function *authenticateAccount* is called. The *authenticateAccount* function creates a query using only the `select` statement. Since only the `select` statement is issued the HPol subject specifies the *selectUser* only has permissions to issue the `select` database command. Figure 6 illustrates the updated PHP code and SQL query enforcing the POLP for the page *login.php*.

Step 3: The modification of the database commands. The appropriate HPol security model for the POLP requires that the action `dbCommands` be broken into individual database commands such as `select_insert`. Subsequently, the newly created action node `select_insert` is further broken into `select` only and `insert` only. This occurs for all database commands allowed in the MySQL database engine.

As an example, returning to the *login.php* page the individual required database command is `select`. In Step 2 the specific least privilege subject *selectUser* was created. In Step 3 the least privilege action `select` is called by the least privilege subject *selectUser*.

Step 4: Modification of the the Mutillidae database tables . The appropriate HPol security model for the POLP requires that the object *star* be restricted to the individual database tables in the Mutillidae web application.

For example, the *login.php* page in Step 2 requires a *selectUser* as the HPol subject. In Step 3 the *login.php* page requires only the database command `select` as the HPol action. In Step 4 the HPol object for the *login.php* page, requires access only to the *accounts* table.

A summary of the POLP as illustrated in Table 2 first displays that for all PHP web pages within the Mutillidae web application the applying least privilege requires only one of three limited database users *selectUser*, *insertUser*, and *selectInsertUser*. Second, these three limited permissions database users can only execute the limited permissions database commands of `select`, `insert`, and `select` and `insert` respectively. Finally, the limited permissions three users, can only execute their limited permission database commands on the database tables of *accounts*, *blogs_table*, and *capture_data*.

5 RELATED WORK

Most of the reported research we found on applying the principle of least privilege focuses on developing new applications while ensuring adherence to the POLP. By contrast, our approach focuses on analyzing and securing existing web applications by building a security model and converting the model and application into POLP compliant implementations.

For the purposes of this short related work section we categorize similar approaches to securing web applications into (1) Least Privilege Models and (2) Secure Web Applications by Design. A more detailed description to other approaches for securing web applications may be found in one of our previous publications [24].

5.1 Least Privilege Models

Wang et al. [26] proposed applying the POLP at authentication time. Elliott and Knight [9, 10] proposed applying POLP during the design process. Jerbi et al. [13] proposed applying POLP to the operating system for hardware protection.

Blankenship and Freedman applied the POLP to develop the Passe system, a replacement for the Django web framework [2]. Passe differs from our work in that it relies on developer-supplied test cases to learn the program flow. In our work we build high-level model of the website security policies and then move this model, and the application, to a least-privilege state.

5.2 Secure Web Application By Design

Galois Inc. [11] developed a secure standalone web server using Haskell. In their approach, web applications are separated from the web server [3, 16]. Also, web applications are built fresh with security built into the application. The Galois approach differs from our research in that our work presumes that a web application has already been built without adequately implementing secure design principles, like most applications currently on the web.

6 CONCLUSION AND FUTURE WORK

6.1 Summary

This paper described the current state of web application security, including the reasons we believe that web applications are unsecured. In order to mitigate the problems with web application security we summarized the concept of the Principle of Least Privilege, and how we can model web application security using a custom tool known as HPol. Combining the concept of the Principle of Least Privilege, the HPol tool and a case study using the Mutillidae web application, we developed the non-least privilege security model for Mutillidae DMBS database permissions. Using the non-least privilege security model, a mechanism was created to move the model from insecure (non-least privilege) to a secure model where least privilege was implemented. A new security model that enforced least privilege was created and this model led to the creation of new PHP code that enforced the principle of least privilege for the Mutillidae DMBS database permissions.

6.2 Ongoing and Future Work

In this paper, we described a formal but manual process for hardening a web application using the principle of least privilege. Our ongoing and future work is focused on three areas of improvement: (1) extracting the security model automatically, (2) converting it to a least privilege model mechanically based on a high-level application access control policy, and (3) refactoring the we application code also automatically.

Since part of this research involve manually implementing the policies for the principle of least privilege, the future work will be automating the process for implementing principle of least privilege policies. The HPol and HERMES frameworks and associated prototypes reported in detail elsewhere [5–7, 14, 15], will be used in future research to automate the process of implementing the principle of least privilege.

ACKNOWLEDGMENTS

We would like to thank all the people that, up to date, have contributed to the HPol and HiFiPol research projects in a variety of roles. Special thanks to Jim Alves-Foss, Matthew G. Brown, Michael Haney, Victor House, Axel W. Krings, Frederick T. Sheldon, Antonius Q. Stalick, Christopher A. Waltrip, and Jared T. Zook.

We would also like to thank the conference chairs, program committee, and reviewers for their time and dedication organizing the CyberSec-2018 conference and reviewing our manuscript. In addition, we would like to thank the funding agencies for partially funding this and prior foundational research. The U.S. National Science Foundation (NSF) under CyberCorps® award 1565572. The State of Idaho under IGEM Cybersecurity Management for Cyber-Physical Systems grant. The opinions expressed in this paper are not those of the NSF or the State of Idaho.

REFERENCES

[1] Brian Barrett. 2016. Most Top Websites Still Don't Use a Basic Security Feature. *Wired Magazine* (mar 2016).

[2] A Blankstein and M J Freedman. 2014. Automating Isolation and Least Privilege in Web Services. In *2014 IEEE Symposium on Security and Privacy*. 133–148. https://doi.org/10.1109/SP.2014.16

[3] David Burke, Joe Hurd, and Aaron Tomb. 2010. High assurance software development. (2010), 29 pages. http://code.galois.com/paper/2010/HighAssuranceSoftwareDevelopment.pdf

[4] Lucian Constantin. 2012. Most of the Internet's top 200,000 HTTPS websites are insecure. *Computer World* (apr 2012).

[5] Daniel Conte de Leon, Venkata A. Bhandari, Ananth A. Jillepalli, and Frederick T. Sheldon. 2016. Using a Knowledge-based Security Orchestration Tool to Reduce the Risk of Browser Compromise. In *Proc. 2016 IEEE 07th Symposium Series On Computational Intelligence (SSCI-2016)*. https://doi.org/10.1109/SSCI.2016.7849910

[6] Daniel Conte de Leon, Matthew G. Brown, Ananth A. Jillepalli, Antonius Q. Stalick, and Jim Alves-Foss. 2017. High-level and Formal Router Policy Verification. *Journal of Computing Sciences in Colleges* 33 (October 2017), 118–128. Issue 1. https://dl.acm.org/citation.cfm?id=3144605.3144631

[7] Daniel Conte de Leon, Paul W. Oman, and Jim Alves-Foss. 2007. Implementation-Oriented Secure Architectures. In *Proceedings of the 40th Hawaii International Conference on System Sciences (HICSS-40)*. IEEE Computer Society, Waikoloa, Big Island, Hawaii, U.S.A., 278a – 278a. https://doi.org/10.1109/HICSS.2007.264

[8] Jeremy Druin. [n. d.]. OWASP Mutillidae 2 Project. https://www.owasp.org/index.php/OWASP_Mutillidae_2_Project

[9] Aaron Elliott and Scott Knight. 2015. Towards Managed Role Explosion. In *Proceedings of the 2015 New Security Paradigms Workshop (NSPW '15)*. ACM, New York, NY, USA, 100–111. https://doi.org/10.1145/2841113.2841121

[10] Aaron Elliott and Scott Knight. 2016. Start Here: Engineering Scalable Access Control Systems. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies (SACMAT '16)*. ACM, New York, NY, USA, 113–124. https://doi.org/10.1145/2914642.2914651

[11] Galois. 2017. Galois. https://galois.com/

[12] Breach Level Index. 2017. Data Breach Database. , 18 pages.

[13] Amir Jerbi, Ethan Hadar, Carrie Gates, and Dmitry Grebenev. 2008. An Access Control Reference Architecture. In *Proceedings of the 2nd ACM Workshop on Computer Security Architectures (CSAW '08)*. ACM, New York, NY, USA, 17–24. https://doi.org/10.1145/1456508.1456513

[14] Ananth A. Jillepalli and Daniel Conte de Leon. 2016. An Architecture for a Policy-Oriented Web Browser Management System: HiFiPol: Browser. In *Proc. 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC-2016)*. https://doi.org/10.1109/COMPSAC.2016.50

[15] Ananth A. Jillepalli, Daniel Conte de Leon, Stuart Steiner, and Frederick T. Sheldon. 2016. HERMES: A High-level Policy Language for High-granularity Enterprise-wide Secure Browser Configuration Management. In *Proc. 2016 IEEE 07th Symposium Series On Computational Intelligence (SSCI-2016)*. https://doi.org/10.1109/SSCI.2016.7849914

[16] John Launchbury. 2007. Cross-domain WebDAV Server. In *Proceedings of the 4th ACM SIGPLAN Workshop on Commercial Users of Functional Programming (CUFP '07)*. ACM, New York, NY, USA, 1 – 2. https://doi.org/10.1145/1362702.1362715

[17] NetCraft. 2017. Total number of Websites. https://news.netcraft.com/archives/category/web-server-survey/

[18] Stack Overflow. 2017. Stack Overflow Developer Survey Results. http://stackoverflow.com/research/developer-survey-2017

[19] OWASP. 2017. Category:OWASP Top Ten Project. https://www.owasp.org

[20] Tutorials Point. 2017. PHP - MySQL Login. https://www.tutorialspoint.com/php/php{_}mysql{_}login.htm

[21] Jerome H. Saltzer and Michael D. Schroeder. 1975. The protection of information in computer systems. *Proc. IEEE* 63, 9 (1975), 1278–1308.

[22] F B Schneider. 2003. Least privilege and more [computer security]. *IEEE Security Privacy* 1, 5 (sep 2003), 55–59. https://doi.org/10.1109/MSECP.2003.1236236

[23] S Sinclair and S W Smith. 2010. What's Wrong with Access Control in the Real World? *IEEE Security Privacy* 8, 4 (jul 2010), 74–77. https://doi.org/10.1109/MSP.2010.139

[24] Stu Steiner, Daniel Conte de Leon, and Jim Alves-Foss. 2017. A Structured Analysis of SQL Injection Runtime Mitigation Techniques. In *Hawaii International Conference on System Sciences (HICSS)*. Kona, HI. https://doi.org/10.24251/HICSS.2017.349

[25] W3Techs. 2017. W3Techs - World Wide Web Technology Surveys. https://w3techs.com/

[26] Hui Wang, Lianzhong Liu, and Wanli Tian. 2012. An authorization model of quantitative analysis of the least privilege. In *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012)*. 283–288.