Information Flow Control for Large Scale Data Repository Architectures

Cameron Merrill*
Michigan State University

August 12, 2015

Abstract

Information Flow Control (IFC) is a new method of data security that focuses on the flow of information between the components of an application. In regards to information privacy, IFC allows for additional security assurance of handling sensitive information beyond the guarantees of just access control and firewalls. Additionally, with online research data repositories with a large number of data sources and users, Information Flow Control allows for greater assurance that information will be handled and transmitted only to authorized sources. This paper goes over the concepts of IFC and their application to the research data repository architecture The Dataverse Project.

1 Introduction

The battle for cybersecurity and secure web based computing has been ongoing since the genesis of the Internet. Furthermore, with the increasing amount highly sensitive data being handled and stored on remote servers, the task of securing information is more relevant than ever. The more lucrative and private the information stored, the more incentive for malicious behavior to try and breach this data. In turn, more incentive means more malicious adversaries to these secure servers holding this sensitive data, which means more attacks and even development of new attacks, leading to an exponential increase in the likelihood of a data breach.

A *data breach* is the intentional (malice) or unintentional (error) release of information to an untrusted environment. A recent report by the Identity Theft Resource Center (www.idtheftcenter.org) estimates the number of data breaches from January 1st, 2015 through the end of July to over 450, with a total number of exposed records being nearly 140 million[1]. Some entities with recent data breaches include the United States Office of Personal Management, with 4.2 million current and former Federal government employee personnel data stolen[2] and even the offensive security firm "Hacking Team" being a victim such a breach[3]

*Work done as part of Harvard University School of Engineering and Applied Sciences REU Program

In modern web security, the most common way of protecting confidential is the mechanism of *access control*. Access control verifies a program or user's access rights at the time and location of access - and grants or denies the entity's access according to the permissions of that entity and the overarching policies of the server. With access control however, no further effort is made to make sure that the accessing entity handles this unlocked information appropriately. Therefore, with access control as the sole mechanism of securing information, an accessing entity may, through or malice, improperly transmit the information in some form. Through this improper transmission of data, confidential information can be pushed to anywhere - potentially including malicious or publicly revealing areas. In short, this means that the mechanism of access control checks placed restrictions on the release of information, but not on the propagation of said data. In large scale computing architectures with many data sources and users, it is unrealistic to assume that all users and data transmissions of data are trustworthy. To ensure that information is only used in accordance with the relevant confidentiality policies, it is necessary to analyze how information flows within the user program - and because of the complexity of these large scale systems, a manual analysis of all possible pipelines of data transmission is unfeasible.

A system can only be deemed "secure" in regards to confidentiality after it has been thoroughly analyzed and proven that any private, authorized information cannot flow to an unauthorized destination. This analysis must show that information that adheres to some policy of privacy, security, or confidentiality, cannot flow to an location where said policy is violated or undermined. These policies are coined as *information flow policies* which are in turn enforced by one or more mechanisms of *information flow control*.

The web application architecture that this paper seeks to implement this framework of information flow control on is that of The Dataverse Project. The Dataverse Project is an open source, large scale data repository web application, that when deployed allows anyone to create their own Dataverse Network. A Dataverse Network acts a hub where researchers can upload, share, explore, cite, and download research data. The Dataverse software is being developed by the Data Science team at Harvard's Institute for Quantitative Social Sciences (IQSS), with the goal of making research data more readily available others, and allowing for other researchers to easily replicate previously done work. This paper will first touch on the policies *information flow control*, and what needs to be enforced for this additional policy assurance, followed by an analysis of the mechanisms utilized to enforce such policies, and finally tying these policies back to the Dataverse Project.

2 Information Flow Control

Information flow control offers a promising approach to security enforcement, where the goal is to prevent disclosure of sensitive data by applications[4]. Several information flow tools have been developed for modern programming languages, such as the Java implementation *Jif*[5], the Caml implementation FlowCaml[6], and the Ada-based *SPARK Examiner*, and many case studies have been done on its effectiveness in security assurance - from security typed languages and cryptographic security [8] to securing web applications via auto-

```
h := h mod 2;
l := 0;
if h = 1 then l := 1
else skip
```

(a) An example of an implicit flow.

```
h := h mod 2;
l := 0;
h := l;
```

(b) A example of an explicit flow.

Figure 1: Examples of the two types of flows information flow controls work to prevent

matic partitioning [9]. With web applications handling more sensitive data while simultaneously becoming more complex to handle more functionality and users, these mechanisms of information flow analysis are becoming very attractive as routes to secure additional assurance for web applications.

2.1 Information Flow Granularity

An important part of Information Flow control and the mechanisms that enforce these policies is the concept of granularity. Granularity refers to the level of which can application, Dataverse in this case, is divided and broken into sub-parts. As IFC is about the monitoring the propagation of data through our system, the amount of subclassification we give our system is a key component, as it is directly relevant to how minutely we monitor these flows of data. Generally, granularity for information flow falls into one of two groups; many subdivisions of an application or architecture into small pieces is classified as *fine granularity information flow control* and the counterpart of fewer subdivisions and larger components is denoted as *coarse granularity information flow control.*

2.2 Implicit and Explicit Information Flows

The two most basic types of information flows are *implicit* and *explicit* flows. When information is passed via a control-structure, it is classified as an *implicit flow*[10]. When information is passed explicitly via assignment from the right-hand of the assignment to the left-hand side, it is classified as an *explicit flow.* Information flow control concentrates on preventing explicit and implicit flows in order to guarantee *noninterference.* Noninterference is the policy that states there is no dependence of public outputs on secret inputs, preventing the leak of secret information through public outputs. Figure 1 shows an example both an explicit and implicit flow in the context of a programming language.

2.3 Static Information Flow Control

Static information flow control is one technique that seeks to protect the privacy and integrity of sensitive data by preventing the unauthorized flow of data. This technique is the method of statically checking information flows within programs that might manipulate the data. Static checking allows the fine-grained tracking of security classes through program computations, with little run-time overhead.

With static information flow control, the running programs that handle data are checked at compile time to gain assurance that all data is handled with provable assurance. One example of such a mechanism would be a type of security typing framework that would be applied to a programming language. This statically typed mechanism would require annotations within the source code that check and prevent secret data from flowing through public outputs via implicit and explicit flows. Static mechanism, such as security typed language frameworks typically fall under the *fine grained information flow control* umbrella, as they break down the flows of information to each implicit and explicit assignment.

2.4 Dynamic Information Flow Control

An alternative approach to static information flow control is the concept of dynamic information flow control. Dynamic information flow control performs dynamic security checks at run-time similar to the checks done by static analysis at compile-time. For example, whenever there is a secret variable on the right-hand sie of an assignment (an explicit flow) or in case the assignment appears inside of a secret (high) conditional statement or while loop (an implicit flow), then the assignment is only allowed in case the assigned variable is designated as having similar high security, or secret level permissions. This mechanism dynamically keeps a simple invariant of no assignment to public variables in secret context.

An example of dynamic mechanism would be a module that monitors the flow of information between endpoints within a given system. With dynamic information flow control, there is potential for more available information to make security decisions due the nature of such a monitor running at run-time instead of compile time - the monitor could have access to information such as the execution trace, or similar sources only available at run-time. This additional information means that dynamic mechanisms have the potential for making more informed decisions at the cost of incurring overhead within our system. These dynamic mechanisms, such as run-time monitors can fall anywhere on the spectrum of information flow granularity, but many lean towards a more coarse grained approach for better performance.

3 The Dataverse Project Architecture

The Dataverse Project is built utilizing the Java Platform, Enterprise Edition - or Java EE for short. The Java EE platform provides an API and runtime environment for developing and deploying enterprise software, such as network and web services and other large-sacle, multi-tiered, scalable, reliable, and secure network applications. Java EE is an extension of the Java Platform, Standard edition (Java SE)[12] - including SE features such as object-relational mapping, distributed and multi-tier architectures, and web services. The platform incorporates a design based largely on modular components running on an application server. The platform emphasizes convention over configuration[11] and annotations for configuration.

The many working components of the Dataverse Architecture make for a complex high level diagram, as seen in Figure 2. The diagram represents how users first interact, or upload data, through the ingest component. The ingest component then stores the relevant parts of the data to the Dataverse Archi-

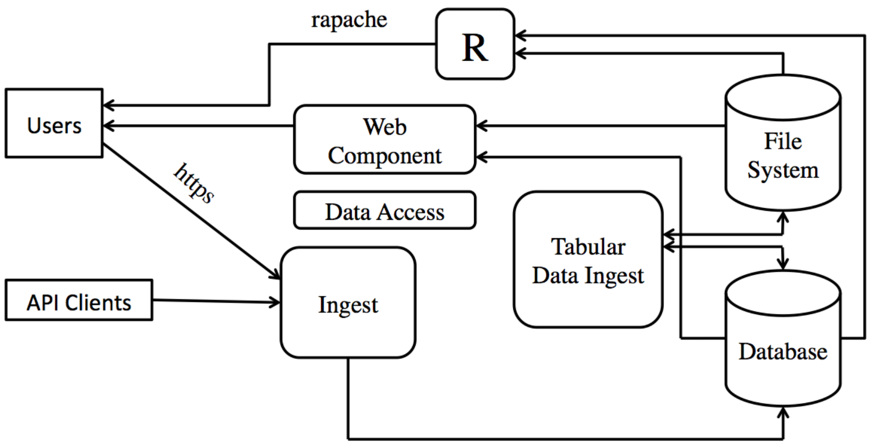

Figure 2: A high level diagram of the Dataverse architecture.

tecture's database or filesystem. From the file system and database, files can later be downloaded or accessed through the web component, then passing them back to users. Additionally, the architecture hosts an asynchronous second level ingest, where some file types are processed as tabular data. The tabular data ingest takes in, or intercepts data from the file system and database, creates archival versions of that data, and then stores the modified version of aforementioned data back onto its respective storage. The components of the file system and database are stored on separate machines than the machine running the Dataverse web application, and utilize closed network communications to pass data. Finally, there are extra components for user functionality, such as the R language web component that pass data back to users. To address these issues, we must look to develop and create a mechanism that incorporates the low overhead of static mechanisms, in a manner that requires as little code rework as possible.

3.1 Enterprise Java Beans

The Dataverse Project utilizes Enterprise JavaBeans (EJBs) within their Java EE architecture. EJB is a Java API for managed, server software and modular construction of enterprise software. An EJB web container provides a runtime environment for web related software components, such as computer security, Java servlet lifecycle management, transaction processing, and other web services. The main idea behind EJB is aiming to provide a standard way to implement server-side, or *back-end*, business software typically found in enterprise applications. EJBs work to handle common concerns such as persistence, transactional integrity, and security in a standard way, leaving software engineers able to concentrate on specific parts of the enterprise software[12].

4 Dataverse and Information Flow Control

The challenge of bringing these concepts of information flow control to the Dataverse architecture is a very complex one. Just glancing at Figure 2 one can see the complexity of the applications architecture at the granularity of the web component level. There are several hurdles that must be overcome in order for this theoretical implementation of information flow control into the Dataverse architecture to be successful.

One issue that must be handled is that of the level of overhaul required in order to incorporate this additional security framework of information flow control. The Dataverse Project has been in development for years, and just released their latest 4.0 version after 18 months of hard work. The implementation of The Dataverse Project is quite large - in the magnitude of hundreds of thousands of lines of code. With this much already existing code, it would unfeasible to design a mechanism of information flow control for Dataverse that required mass code rework. For this mechanism to be successful, it must be as *"plug and go"* as possible, with little adaptation from the Dataverse required in order to make things work. This rules out the potential for static mechanisms of information flow control, such as the previously mentioned statically typed security system - going through every line of code and adding annotations is not an acceptable approach.

A second issue that must be addressed is the performance of the Dataverse web application with this new mechanism implemented. As a static mechanism is ruled out, the most plausible next option would be a dynamic mechanism, some form of run-time process that enforces these information flow policies that satisfy our security demands. As previously discussed, such a mechanism at run-time has a potential for occurring overhead, and therefore could negatively impact end-user experience of the application.

To combat these issues, we are steered towards an implementation that attempts to bridge the benefits of both *static* and *dynamic* mechanisms; the performance benefits of static mechanisms, with the minimal code adaptation of dynamic run-time monitors. These requirements of a successful implementation lead us to a new type of information flow control - *Decentralized Information Flow Control.*

4.1 Decentralized Information Flow Control

Decentralized Information Flow Control aims to bridge the gap between the models of precise information flow control and its ease of implementation (but high overhead) with the models of precise characterization of information flows and their lack of overhead (but challenging implementation). The decentralized model of Information Fow Control addressed the weakness of earlier approaches in protecting confidentiality in a system with many users, even in scenarios of mutual distrust.[13] The Decentralized Information Flow Control (DIFC) model allows users to control the flow of their information without imposing the rigid constraints of a traditional multilevel security system. All practical information flow control systems provide the ability to arbitrarily weaken information flow restrictions (or "declassify" data), as otherwise these information flow controls would be too restrictive for writing real user based applications. With the DIFC model, this *declassification* is performed differently than past implementations;

it is performed by a *trusted process*, which possesses the authority of a universally trusted principal. The Flume[14] model for DIFC applies these concepts at the granularity of operating system processes and standard OS abstractions, and serves as the basis for the Dataverse implementation of information flow controls.

5 Decentralized Information Flow Control Model

The Flume[14] model for DIFC applies these concepts at the granularity of operating system processes and standard OS abstractions, and serves as the basis for the Dataverse implementation of information flow controls. This model makes for ease of use for DIFC in existing applications, such as The Dataverse Project - the implementation at operating system process granularity means this model, a modified Linux operating system, can run existing software with minimal changes to incorporate the security assurance of information flow controls. The Dataverse DIFC implementation integrates common DIFC control mechanisms with standard communication abstractions of operating systems - pipes, sockets, and file descriptors, through a user level reference monitor.

5.1 Process and File Labels

This model of DIFC uses tags and labels to track data as it flows through a system. Let T be a very large set of opaque tokens called tags. A tag t carries no inherent meaning, but processes generally associate each tag with some category of secrecy or integrity. Tag b, for example, might label Bob's private data.

Labels are subsets of T. Labels form a lattice under the partial order of the subset relation. Each process p has two labels, S_p for secrecy and I_p for integrity. If tag $t \in S_p$, then the system conservatively assumes that p has seen some private data tagged with t. A process whose secrecy label contains one or more tags requires independent consent for each tag to reveal data publicly. For integrity, if $t \in I_p$, then every input to p has been endorsed as having integrity for t. Files (and other objects) also have secrecy and integrity labels.

Although any tag can appear in any type of label, in practice secrecy and integrity usage patterns are so different that a tag is used either in secrecy labels or in integrity labels, not both. We therefore sometimes refer to a "secrecy tag" or an "integrity tag".[14]

5.2 Decentralized Privilege

In standard centralized Information Flow Control, the process of creating new tags, subtracting tags to secrecy labs (*declassifying information*) and adding tags to integrity labels (*endorsing information*) is typically done by a sole trusted security administrative process. However in this model, we allow any process the ability to create new tags, which in turn gives the process the ability to declassify and endorse information for that specific created tag.

In this system, each tag has two related capabilities; for a given tag t, our system ties the t^+ and t^- capabilities to the tag. These respectively represent the ability for a process within our system to add (t^+) or subtract (t^-) that specific tag. Every process in our system owns a set of capabilities, represented as O_p, for arbitrary process p. If $t^+ \in O_p$, then we say that process p as the ability to add the t tag to it's secrecy (S_p) or integrity (I_p) labels. The ability to add a tag t, or the capability t^+ lets a process grant itself the ability to receive data labeled with data t, while t^- allows it declassify any secret t it has seen via removing the t tag from it's secrecy label.

Any process in our system can allocate, or create a completely new tag. When process p creates a new tag t, our systems sets $O_p \leftarrow O_p \cup \{t^+, t^-\}$, meaning process p now has the capability of adding and subtracting t. When a process has both the ability to add and subtract a specific label, or when $\{t^+, t^-\} \subseteq O_p$, we say it has *dual privilege*. The set $D_p = \{t | t^+ \in O_p$ and $t^- \in O_p\}$ represents for the set of all tags that process p has dual privilege for.

5.3 Security

For our model, and in the Dataverse theoretical implementation, we assume a system of many processes running on the same machine communicating via messages, or "flows". The goal of this model is to track data by regulating process communication and process label changes.

Definition 1. A system is defined as secure in regards to this model if and only if all allowed process label changes are "safe" (Definition 2) and all allowed messages are "safe"(Definition 3).

Safe Label Changes In this model, only process p itself can change its own labels, S_p and I_p. In order to make a change, process p must request a change from the overarching reference monitor. The reference monitor, a trusted process, allows said change if it follows Definition 2.

Definition 2. For a process p, let L be S_p or I_p, and let L' be the new value of the label. The change from L to L' is *safe* if and only if:

$$\{L' - L\}^+ \cup \{L - L'\}^- \subseteq O_p$$

For this definition, we restrict our label changes to only be those for which a given process has the correct capability for. Any change in label for process p that results in the addition of tag t is only allowed if $t^+ \in O_p$

Safe Messages Information flow control restricts process communication to prevent data leaks. This model restricts communication among unprivileged processes as in classical IFC: p can send a message to q only if $S_p \subseteq S_q$ and $I_q \subseteq I_p$. If two processes in our system could communicate by changing their labels, sending a message using the centralized rules, and then restoring their original labels, then model can safely allow the processes to communicate without label change. These temporary label changes are allowed for a process has dual privilege for the tag(s) it wishes to temporarily add or remove.

Definition 3: A message from p to q is safe if and only if:

$$S_p - D_p \subseteq S_q \cup D_q \text{ and } I_q - D_q \subseteq I_p \cup D_p$$

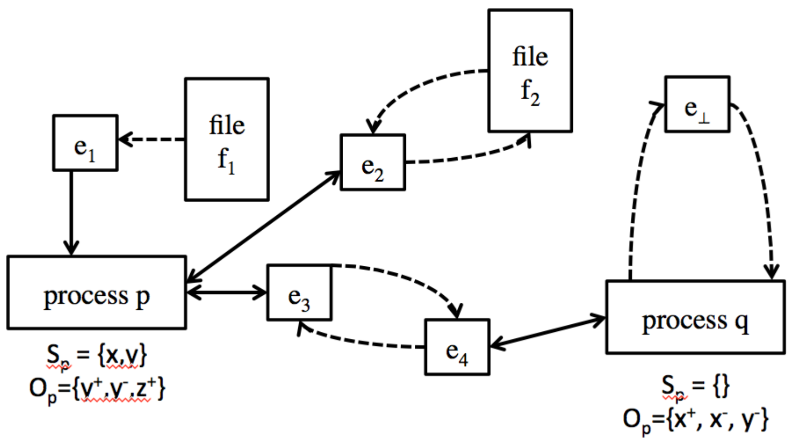

Figure 3: An example diagram of endpoints and labels in use within a system.

5.4 Endpoints

A challenge for this model and the incorporation of existing communication interfaces such as sockets and pipes is how to accommodate processes these abstractions and how and when they use their privileges. The solution implemented within this model of DIFC is that of *endpoint abstraction.* Each resource a process uses to communicate is represented as an endpoint (pipes, sockets, files, network connections, etc). A process can than designate permissions on each of the communications through these endpoints.

When a process p acquires a new file descriptor, it gets a new corresponding *endpoint.* Each endpoint e has its own secrecy and integrity labels, S_e and I_e. By default, $S_e = S_p$ and $I_e = I_p$. A process owns readable endpoints for each of its readable resources, writable endpoints for writable resources, and read/write endpoints for those that are bidirectional. Endpoints meet safety constraints as follows:

Definition 4. A readable endpoint e is safe if and only if:

$$(S_e - S_p) \cup (I_p - I_e) \subseteq D_p$$

A writable endpoint e is safe if and only if:

$$(S_p - S_e) \cup (I_e - I_p) \subseteq D_p$$

With endpoints, now all process communication now happens between two endpoints, not two processes, requiring a new Definition 3.

Definition 5. A message from endpoint e to endpoint f is safe if and only if e is writable, and f is readable; $S_e \subseteq S_f$ and $I_f \subseteq I_e$.

We can now prove that any message between two safe endpoints is also a safe message between the endpoints that process corresponds to. Take process p with safe endpoint e, process q with safe endpoint f, and a safe message e to

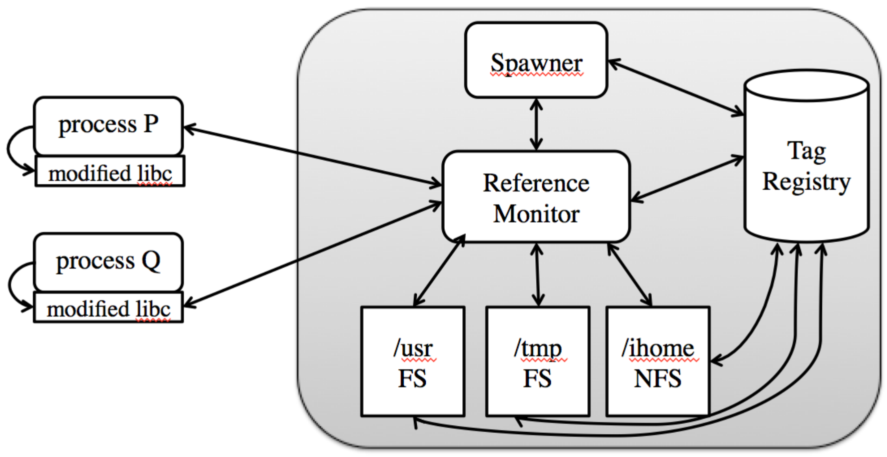

Figure 4: High-level design of the DIFC implementation. The shaded area represents the trusted computing base.

f. In terms of secrecy, that the message between the endpoints is safe implies by Definition 5 that e is writable, f is readable, and $S_e \subseteq S_f$. Since e and f are safe, Definition 4 implies that $S_p - D_p \subseteq S_e$ and $S_f \subseteq S_q \cup D_q$. Thus, $S_p - D_p \subseteq S_q \cup D_q$, and the message between the processes is safe for secrecy by Definition 3. A similar argument holds for integrity.

6 Decentralized Information Flow Control Implementation

While this model of Decentralized Information Flow Control has not been implemented in the Dataverse architecture yet, it has been done for similar web applications and can be easily extrapolated to the Dataverse architecture. This section covers those similar implementations so that future researchers of this project could easily pick up on.

The implementation of the previously outlined conceptual model is done via a user-space modification of Unix. By designing this mechanism for the user-space, this means that the application has exceptional portability and ease of implementation. From these aspects, there is potential that this form mechanism could be bundled with The Dataverse Project for networks other than the Harvard Dataverse Network. The Linux implementation, which is similar to Ostia[15] and Plash[16] as a user space design for confinement, uses a small kernel level component: a Linux Security Module, or LSM, which implements system call interposition[17]. Figure 4 displays the major components of the implementation. The *reference monitor* (RM) keeps track of each process' labels, authorizing or denying requests to change labels. Additionally, the reference monitor handles system calls for the processes. The reference monitor relies on several helpers: a remote tag registry, user space file servers, and a modified C library that is aware of the DIFC implementation. The modified C library allows for Unix system calls to be redirected to the RM and also allows for the creation of new API calls for our system. The below list shows a few examples of new API calls.

- *label get_label*$(\{S, I\})$
  Return the current process's S or I label
- *capset get_ownership*$()$
  For the current process p, return the capability set O_p.
- *int change_label*$(\{S, I\},$ *label l*$)$
  Set current process's S or I label to l, so long as the change is safe (Defn 2.) nad the change keeps all endpoints safe (Defn 4). Returns error code on failure.
- *label get_fd_label*$(\{S, I\},$ *int fd*$)$
  Get the S or I label on file descriptor $fd's$ endpoint
- *tag create_tag*$(\{EP, IP, RP\})$
  Create a new tag t for the specified security policy (export, integrity, or read protection). In the first case, add t^+ to O_p; in the second add t^- to O_p; and in the third add neither.
- *pid spawn*$(char\ *argv[], char\ *env[], tokenpipes[],$ $[$*label S*, *label I*,-*capset O*$])$
  Spawn a new process with the given command line environment. Collect given pipes. By default, set secrecy, integrity, and ownership to that of the caller. If S, I, and O are supplied and represent a permissible setting, set labels to S, I, and ownership to O.
- *int claim_fd_by_token*$($*token t*$)$
  Exchange the specified token for its corresponding file descriptor.
- *int pipe*$(int\ *fd, token\ *t)$
  Make a ne pipe, returning file descriptor and pipe token.

6.1 WikiTest Implementation

This section explores the implementation of our system within a popular web publishing system, MoinMoin wiki. MoinMoin wiki allows serves as a test-run for these concepts and implementations on a smaller scale of web application compared to The Dataverse Project. MoinMoin is a popular Python-based web publishing system - a wiki - that allows web clients to read and modify server hosted pages. MoinMoin is designed to share documents between each users, but each page can have its own access control list which governs which users (and groups) have the ability to modify it.

What makes MoinMoin a good candidate for our information flow control security framework is that it has notoriously been a source of security problems. Moin is comprised of roughly 91,000 lines of code across 349 modules. Moin's ACL (access control list) are checked in 41 places across 22 different modules, writing ACLs in 19 places among 12 modules. The issue with Moin's ACL is that a check could be easily omitted. A public vulnerability and MoinMoin's internal bugtracker[19][20] show at least five recent ACL-bypass vulnerabilities.

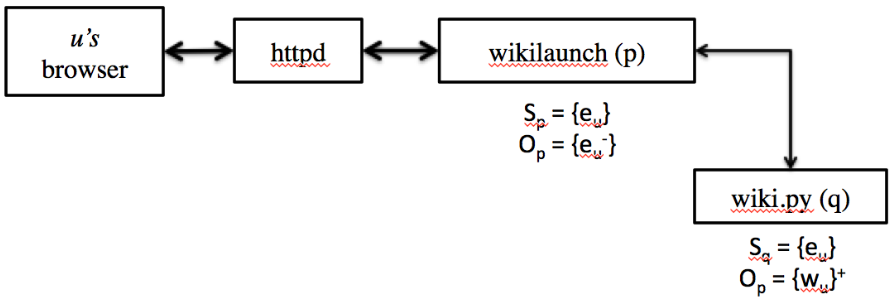

Figure 5: Label setup for read or write request in TestWiki. wiki.py only gets capability w_u^+ if writing. The target page is export and write protected by user u.

These ACL bypass vulnerabilities are security flaws that we attempt to fix by incorporation our DIFC system into MoinMoin's implementation.

Our DIFC system's approach to enhancing MoinMoin's read and write protection is to factor our security code into a small, isolated security module, and leave the rest unchanged. This parallels the theoretical desired implementation of Dataverse, making this MoinMoin example a suitable trial run for our DIFC system. This factored out security code configures and performs the DIFC policies, running our application (MoinMoin) in regards to these policies.

Our MoinMoin implementation, denoted as TestWiki, uses an unmodified Apache Web server (httpd) for front-end request handling. Figure 5 shows the high level view of the TestWiki main components. *wiki.py* is the majority of the application code, which is mostly untouched from the standard MoinMoin application. *pmgr.py* is a small trusted program that manages usernames and passwords; it runs a setlabel program so that it may compare submitted passwords against read-protected hashes on the server. *wikilaunch* is the small trusted security code module; it is responsible for interpreting the Web request, launching *wiki.py* with the correct DIFC policy, and proxying *wikilaunch*'s response back to Apache.

6.2 TestWiki Evaluation

The two aspects we look to review of the TestWiki implementation of our DIFC model are the factors of security and performance. In this dummy implementation, we find that our model prevents ACL vulnerabilites and even helps discover new ones. For performance, we find that our mechanism adds from 35-286 microseconds of overhead to interposed system calls - an insignificant change. At the system level, the throughput and latency of our system adds between 35-45% of overhead in comparison to the unmodified MoinMoin implementation.

The most important evaluation criterion for Flume is whether it improves the security of existing systems. Of the five recent ACL bypass vulnerabilities [25, 26], three are present in theMoinMoin version (1.5.6) we forked to create FlumeWiki. One of these vulnerabilities is in a feature disabled in FlumeWiki. The other two were discovered in code FlumeWiki indeed inherits from Moin.

We verified that FlumeWiki still "implements" Moin's original buggy behavior and that the Flume security architecture prevents these bugs from revealing private data.

To evaluate the system level performance overhead of our mechanism, we compare the throughput and latency of pages served by an unmodified Moin-Moin wiki and by TestWiki.TestWiki is 43% slower than MoinMoin in read throughput, 34% slower in write throughput and it adds a latency overhead of roughly 40ms. For both systems, the bottleneck is the CPU. Moin- Moin spends most of its time interpreting Python and TestWiki has the additional system-call and IPC overhead of Flume.

The TestWiki implementation of the DIFC model was successful in adding additional security at the level of the user, solving old issues and remedying unidentified ones. Further work can be done on this mechanism and model to achieve better performance - the web response and system calls do not see much overhead, but there is always room for improvement, especially with server latency throughput and latency. From the success of this implementation, it is provably seen that we now have a mechanism that meets the needs of the Dataverse Project; minimal intrusiveness of the mechanism, additional security, and manageable overhead.

7 Conclusion

The model of Decentralized Information Flow Control combined with the Flume implementation and MoinMoin example demonstrate that the advantages of DIFC can be brought to standard operating systems and web applications with minimal overhaul on the application side. The modified Linux operating system allows programmers a strong assurance that applications will behave securely, even with untrusted code.

Through the results of the MoinMoin experiment of this model of DIFC, it can readily seen that this model and mechanism combination are successful in bringing DIFC policies through the operating system medium. Such a mechanism meets the original goals outline for this project; to find an implementation that would require minimal overhaul to the Dataverse Project code and would provide additional security assurance. The Dataverse Project architecture may be a complicated one, but the nature of this mechanism and the operating system medium allows this model to work through these challenges and still proves to be a viable potential extension to Dataverse.

This new framework of *information flow controls* is the solution to any and all solutions regarding of securing information, but it is certainly a further step in the battle for cybersecurity. This framework aims to pick up where access control falls through, and has the potential to serve as a very valuable addition to the Dataverse Project and Harvard's Dataverse Network.The code and additional resources are available for this project via contacting the author.

8 Acknowledgements

The author would like to thank the entirety of the Privacy Tools for Sharing Research Data group at Harvard University for this opportunity to learn and grow as a Computer Scientist. He would like to thank his mentors for the duration of this project, Stephen Chong and Gustavo Durand, for their guidance and insight offered. The author would also like to thank Salil Vadhan for his support over the summer and for providing this opportunity. The author would like to extend his gratitude towards all of his fellow DP interns, as well as fellow REU interns Julia Redston and Emily Engel for their friendship and support through many long days. Finally, the author would like to express his endless gratitude to his family for their constant support, encouragement, and words of wisdom throughout the project.

References

[1] Identity Theft Resource Center. "2015 Data Breach Reports" TRC, 4 August 2015. Web. 10 August 2015.

[2] Raston, Dina Temple. "U.S. Officials Say Nearly 14 Million Affected In OPM Breach." NPR. NPR, 15 June 2015. Web. 07 July 2015.

[3] Schwartz, Matthew J. "Surveillance Software Firm Breached." *Data Breach Today.* Kroll Advisory Solutions, 6 July 2015. Web. 6 July 2015.

[4] A. Sabelfeld and A. C. Myers. Language-based information flow security. *IEEE J. Selected Areas in COmmunications*, 21(1):5-19, Jan. 2003.

[5] A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. Jif: Java information flow. Software release. Located at http://www.cs.cornell.edu/jif, July 2001-2008

[6] V. Simonet. The Flow Caml system. Software release. Located at http://cristal.inria.fr/simonet/soft/flowcaml, July 2003.

[7] J. Barnes and J. Barnes. *High Integrity Software: The SPARK Approach to Safety and Security.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[8] A. Askarov and A. Sabelfeld. Security-typed languages for implementation of cryptographic protocols: A case study. *In Proc. European Symp. on Research in Computer Security,* volume 3679 of LNCS, pages 197–221. Springer-Verlag, Sept. 2005.

[9] S. Chong, J. Liu, A. C. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. *In Proc. ACM Symp. on Operating System Principles*, pages 31–44, Oct. 2007.

[10] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM,* 20(7):504–513, July 1977.

[11] ZEEF. "J-Development." *JDevelopment RSS.* N.p., 21 Aug. 2011. Web. 11 Aug. 2015.

[12] Oracle, Inc. "Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform." *Java EE Documentation.* Oracle, 2012. Web. 11 Aug. 2015.

[13] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Computer Systems,* 9(4):410–442, October 2000.

[14] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. 2007. Information flow control for standard OS abstractions. *SIGOPS Oper. Syst. Rev. 41, 6 (October 2007),* 321-334. DOI=10.1145/1323293.1294293 http://doi.acm.org/10.1145/1323293.1294293

[15] T. Garfinkel, B. Pfaff, and M. Rosenblum. Ostia: *A delegating architecture for secure system call interposition.* In Proc. 2004 NDSS, February 2004.

[16] M. Seaborn. Plash: tools for practical least privilege.

[17] C.Wright,C.Cowan,S.Smalley,J.Morris,and G. Kroah-Hartman. Linux security modules: General security support for the Linux kernel. *In Proc. 11th USENIX Security,* Aug. 2002.

[18] N. Provos. Improving host security with system call policies. *In Proc. 12th USENIX Security,* Aug. 2003.

[19] National Vulnerability Database. CVE-2007-2637. http://nvd.nist.gov/nvd.cfm?cvename=CVE-2007-2637.

[20] osvdb.org. Open Source Vulnerability Database. http://osvdb.org/searchdb.php?base=moinmoin. http://plash.beasts.org.