Moving Target Defense for Web Applications

by

Marthony Taguinod

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2017 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Co-Chair
Adam Doupé, Co-Chair
Sik-Sang Yau

ARIZONA STATE UNIVERSITY

May 2018

ABSTRACT

Web applications continue to remain as the most popular method of interaction for businesses over the Internet. With it's simplicity of use and management, they often function as the "front door" for many companies. As such, they are a critical component of the security ecosystem as vulnerabilities present in these systems could potentially allow malicious users access to sensitive business and personal data.

The inherent nature of web applications enables anyone to access them anytime and anywhere, this includes any malicious actors looking to exploit vulnerabilities present in the web application. In addition, the static configurations of these web applications enables attackers the opportunity to perform reconnaissance at their leisure, increasing their success rate by allowing them time to discover information on the system. On the other hand, defenders are often at a disadvantage as they do not have the same temporal opportunity that attackers possess in order to perform counter-reconnaissance. Lastly, the unchanging nature of web applications results in undiscovered vulnerabilities to remain open for exploitation, requiring developers to adopt a reactive approach that is often delayed or to anticipate and prepare for all possible attacks which is often cost-prohibitive.

Moving Target Defense (MTD) seeks to remove the attackers' advantage by reducing the information asymmetry between the attacker and defender. This research explores the concept of MTD and the various methods of applying MTD to secure Web Applications. In particular, MTD concepts are applied to web applications by implementing an automated application diversifier that aims to mitigate specific classes of web application vulnerabilities and exploits. Evaluation is done using two open source web applications to determine the effectiveness of the MTD implementation. Though developed for the chosen applications, the automation process can be customized to fit a variety of applications.

To my parents

Acknowledgements

I would first like to express my deepest gratitude to my co-chairs Dr. Gail-Joon Ahn and Dr. Adam Doupé for providing me the opportunity to work on various cutting edge research projects, as well as providing motivational support and guidance in both educational and personal growth throughout my graduate career. I would like to extend my gratitude to Dr. Sik-Sang Yau for serving on my committee and providing valuable feedback on my thesis. My time at the laboratory for Security Engineering for Future Computing (SEFCOM) has allowed me to connect with innovative and highly motivated individuals, as such, I consider it an honor working with them and am extremely grateful for their advice and motivational support. Special thanks to Faris Bugra Kokulu for his assistance in completing various experiments. Lastly, and most importantly, I would like to extend my sincere love and gratitude to my parents and brother for being a constant source of motivation and support throughout my academic career.

This work would have not been possible without the financial support from the National Science Foundation, Scholarship for Service (NSF-SFS-1129561).

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Static System Configurations: Conventional System Configurations

Current approaches to security assume that a systems' configuration remains static over long periods of time. This enables the attacker to have two key advantages over the defender: The ability to perform system reconnaissance over a period of time undetected and the ability to develop exploits based on the information gathered and execute them on their own time. Moving Target Defense seeks to remove this advantage by changing system configurations over time. Chapter 2 discusses MTD further, providing background and related work.

Web applications continue to be the most widely used method for businesses to conduct services over the Internet. Too often, sensitive business and user data is managed and processed by these web applications. As such, vulnerabilities in these systems pose a serious threat to the confidentiality, integrity, and availability of business and user data. Existing tools and techniques focus on identifying and preventing these vulnerabilities. However, due to the increasing complexity of modern web applications and their slow deployment time, they are often ineffective. To address these issues, Moving Target Defense can be leveraged in order to provide another layer of defense. Chapter 3 further discusses the modern web application architecture and its vulnerabilities leading to Chapter 4, where a discussion of current techniques followed by an exploration in related work of Moving Target Defense in web applications is done.

1.2 Research Approach

A study was done on current Moving Target Defense techniques in order to gain an understanding of its goals and challenges and identify the composition of Moving Target Defense (What to Move, When to Move, Limitations, and evaluation techniques). In order to develop a Moving Target Defense system applicable to Web Applications, we dissect the structure of web application to identify its components and the vulnerabilities possible for each component. Doing so allows for identification of configurations that can be 'moved', in addition to identifying the limitations of applying Moving Target Defense in specific configurations. Furthermore, some work was done in designing effective movement policies in order to maximize the security of web applications

1.3 Limitations

The proposed technique is applicable to any language, but since we utilize application-specific 'shims' that have to be developed manually, scalability issues might arise depending on the amount of 'shims' to create. A collection of these existing 'shims' might alleviate this problem but having such application specific code publicly available renders the technique ineffective as malicious users would also have the same information available to them. The proposed method is also limited in prevention of specific attacks and would best be used with other defense techniques in a layered approach.

1.4 Contributions

In summary, the main contributions expected to be explained in this work are:

- A generalized MTD framework that can be applied to any web application to

any layer

- An implementation of an automated language and database translator that enables source and dialect randomization for web applications.
- Application of MTD to open source web applications using source and dialect randomization and their effectiveness.

Chapter 2

MOVING TARGET DEFENSE BACKGROUND

2.1 Introduction to Moving Target Defense

Moving Target Defense (MTD) seeks to level the asymmetric environment of attacker and defender by negating any advantages the attacker has. In order to accomplish this goal, MTD seeks to control change across various system dimensions to increase uncertainty, complexity, and cost for attackers Cyberspace (2011). With the assumption that perfect security is difficult to obtain, MTD focuses on enabling resilient, defensible systems that allow continued, safe operation in a compromised environment rather than developing perfectly secure systems. In addition, MTD does not remove vulnerabilities directly, rather, it reduces the attack window by limiting the temporal exposure of vulnerabilities.

Two approaches in creating such a defensible system are: disrupting reconnaissance efforts of the attacker or disrupting on-going attacks. For both MTD approaches, they aim to invalidate any information on the system that an attacker had managed to acquire previously.

2.2 Challenges in Moving Target Defense

There are several main challenges in developing efficient and effective MTD systems:

- Ensuring that legitimate users have continued access to the service
- Ensuring that the MTD technique introduces minimal cost (in terms of delay and resource consumption) to the system

- Ensuring that the MTD technique provides benefit when introduced to the system

2.3 Related Work

The idea and philosophy of MTD, which is to increase uncertainty and complexity for attackers, has been proposed and studied for decades Avizienis and Chen (1977); Ammann and Knight (1988); Pettis and Hansen (1990); Forrest *et al.* (1997).

Okhravi *et al.* surveyed techniques that applied the philosophy of MTD in different cyber research domains Okhravi *et al.* (2013). According to them, existing techniques can be categorized into five classes based on what component to move:

1. Changing the application environment Team (2003); Barrantes *et al.* (2003)
2. Changing application code dynamically or diversifying software Wartell *et al.* (2012); Larsen *et al.* (2014)
3. Changing the representation of data Ammann and Knight (1988); Nguyen-Tuong *et al.* (2008)
4. Changing the properties of platforms Williams *et al.* (2009); Salamat *et al.* (2011)
5. Changing the network configurations Zhuang *et al.* (2013); Ge *et al.* (2014); Jafarian *et al.* (2012)

Application environment randomization involves modifying the environment presented to the application by the system at run-time. These techniques modify configuration components such as data and instruction memory locations, heap/stack configuration, and the application's instruction set. Techniques that fall within this category typically prevent injection-attacks that seek to control the application by inject-

ing malicious code or otherwise. Address Space Layout Randomization (ASLR) Team (2003) and Instruction Set Randomization (ISR) are widely adopted instances of application environment randomization in modern operating systems. Existing ASLR mechanisms randomly arrange the address space positions of key data areas such as the base executable memory location, application stack and heap, and any libraries it requires. (what-to-move) of a process when it is launched (when-to-move), including the base of the executable and the positions of the stack, heap, and libraries. As a result, if an attacker manages to exploit some memory corruption vulnerability in the application binary, i.e. a buffer overflow attack, it would be difficult for attackers to transfer control flow to their injected code as they will be unable to accurately predict the application's memory layout.

Dynamic application code or code diversification involves techniques that change the application code dynamically, for instance techniques include modifying project instructions or having different, multiple versions of the application (hence diversity). The authors in Okhravi *et al.* (2013) cite Proactive Obfuscation as one example technique wherein they create multiple copies of the service randomized differently - that is, semantically different but functionally equivalent applications. Whenever a request to the service is issued, it is sent to each service replica and responses from each are calculated and the majority vote is the response that is sent out.

Dynamic data representation is similar to dynamic application code or code diversification, but instead of having multiple variants of the application code being analyzed, multiple variants of the application data is monitored instead. It does so by having multiple copies of the application wherein each replica handles the data passed differently - any variants or divergence among the services is then detected and alerts the administrator of malicious behavior.

Dynamic platform techniques involve any change within the platform properties

of the system - OS version, CPU Architecture, OS Instance, etc. Multi variant Execution is one example of this technique wherein multiple varieties of the service is executed in lockstep. While the applications run identically in normal behavior, each service replica operates differently while under attack. These inconsistencies are then monitored and reported once detected in order to signify an ongoing attack.

Dynamic network techniques primarily involve changing the properties of the network dynamically, some examples include protocols, ports, and addresses. Some well known techniques include IP Randomization Jafarian *et al.* (2014, 2012) and Port Randomization Luo *et al.* (2014). Dynamic network techniques often focus on disrupting the reconnaissance effort of the attacker in that the IP address, traffic, or open ports revealed are changed in order to delay their information gathering.

In all five of their proposed categories, each technique involves identifying anomalous behavior and alerting the user or administrator of the suspected activity. However, determining if the action taken is anomalous or not is difficult due to the possibility of false positives and false negatives, compounded by the unpredictable and ever-changing actions of human attackers. Therefore their exists the research challenge of developing an effective intrusion detection and anomaly detection system to address the dynamically changing actions of attackers in order to separate legitimate traffic from suspicious and malicious traffic.

2.4 Dynamic Network Techniques

Research in the area of Network-Based MTD techniques are plentiful and involve techniques that modify the end-host communications to techniques that modify the actual network infrastructure. As with any approach, they all involve trade-offs and challenges. In network-based MTD, the difficulty of identifying legitimate and malicious users become more prominent as traffic sent by both are similar to each other -

malicious actions being easily identified during the attack phase. In addition, introducing a MTD technique to the system may inadvertently cause a bottleneck, creating an unintended weak point for Denial of Service attacks - denying legitimate users access to the service itself. Finally, the problem of scalability becomes a greater focus due to constantly changing business requirements and the varying network requirements of each organization. There are five general categories that existing network based MTD techniques can fall under: Port Randomization, Traffic Morphing, Dynamic Network Address Translation, and Network Address Space Randomization.

2.4.1 Port Randomization

Port Hopping Luo *et al.* (2014, 2015) is a technique that constantly modifies the port number in order to prevent reconnaissance attacks on a service. It allows one to hide service identities thereby confuses attackers to the real location of each service. However, the data contained in relevant work shows that Port Hopping is effective mostly in systems that contain few vulnerabilities and a large number of ports. In addition, port hopping seems to be only effective during the reconnaissance phase of attacks, as once an attacker manages to get in the system, changing the port would not matter as much due to being able to keep track of changes.

2.4.2 Traffic Morphing

Traffic Morphing Li *et al.* (2014) is a technique that involves hiding the intended packet within typical network traffic in order to prevent successful traffic analysis. The technique is primarily used by Cyber-Physical Messages in order to satisfy the real-time constraints of Cyber Physical Systems. Experimental results of the research reveal that the proposed technique achieves moderate overhead to the system while successfully morphing the messages with typical network traffic during real-time.

2.4.3 Dynamic Network Address Translation

Dynamic Network Address Translation Kewley *et al.* (2001); Michalski *et al.* (2002) works by hiding/obfuscating the hosts' identity located with each packet header when it is sent over the public Internet. This delays and introduces confusion to the attacker by hiding what is occurring within the network, however not all applications can work with address translation, in addition to the overhead costs that get added to each connection.

2.4.4 Network Address Space Randomization

Network Address Space Randomization Antonatos *et al.* (2005) primarily involves hosts forcibly changing their IP addresses by requesting to requesting a new address from the DHCP server. This technique is supported by different desktops and operating systems. However, the technique requires constant resource requests and changes to the OS on each end host, thereby increasing the deployment cause.

To address the issue of service availability to legitimate users, Crosb et. al. Crosby *et al.* (2013) analyzed the the network interdependencies of Moving Target Defense Systems and proposed a layered approach in viewing interdependencies between layers in order to develop tools and techniques that allow continuous service to legitimate users. In their layered approach, they suggested to make 'corrections' or changes to the same layer or a higher layer where the randomization occurs to perform mitigation.

For example in Figure 2.1, if IP randomization, Layer 3, was the chosen MTD technique to deploy, the mitigation strategy for legitimate users should be done at a higher level such as Layer 7. At the same time however, any changes done at any layer might have an effect on the lower layer dependencies.

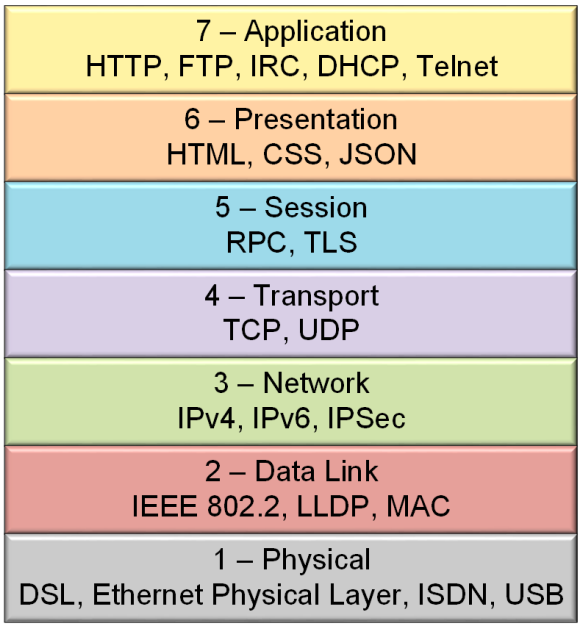

Figure 2.1: OSI Layers

In order to support the idea of network-based MTD solutions, Ge et. al. Green *et al.* (2015) proposed a generic framework for MTD by introducing a service layer between users and servers in order to avoid exposure of actual servers. Figure 2.2 shows their proposed MTD framework, wherein a proxy server authenticates users and assigns each legitimate user to a proxy node. The proxy node then maps and relays communication between the users and the actual servers. In this way, the servers true identity is not revealed to the users due the proxy nodes functioning as communication relays. In addition, only authenticated users would be allowed access to the proxy nodes, reducing the chance of unauthorized access to the services.

They also identify several seven key properties that are common to network based MTD techniques that ensures their effectiveness in addition to identifying common components among the approaches. According to them, there are four main components in network-based MTD:

- Clients are hosts that are trying to access the service that is protected by the

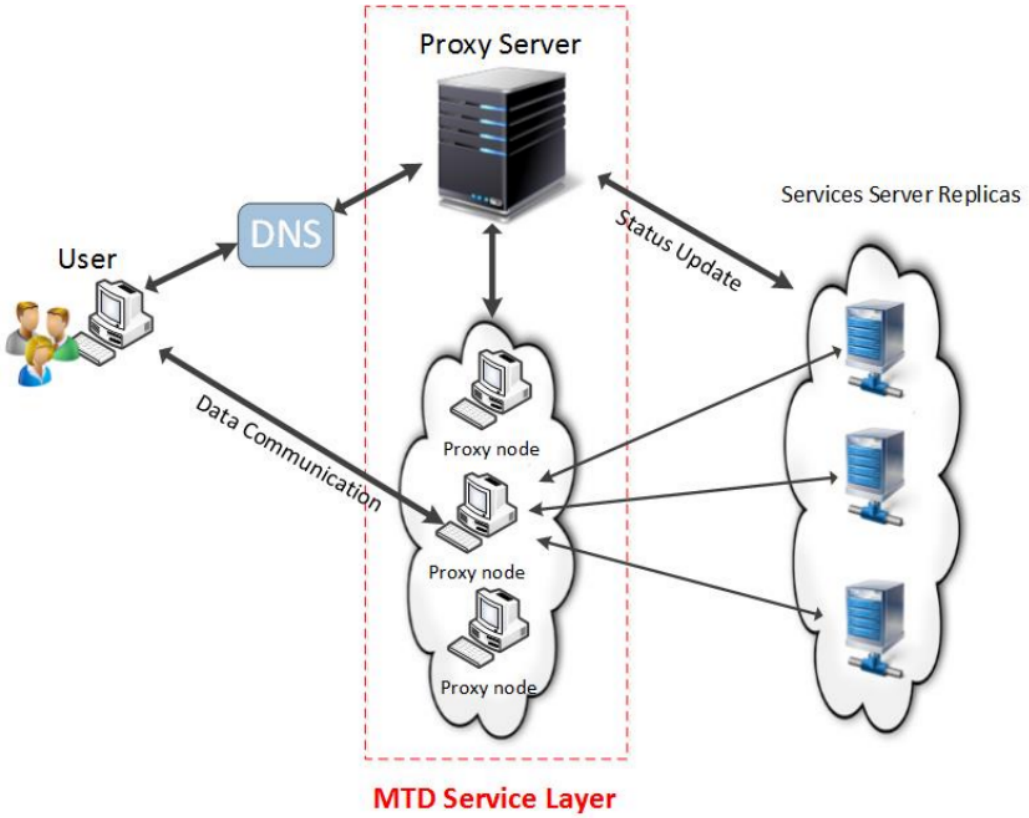

Figure 2.2: MTD Framework proposed by Ge *et al.* (2014)

MTD system. They are divided into two types:

- Trusted Clients are clients that are not deemed malicious and follow typical actions toward the service.
- Untrusted Clients are clients that have not been granted access to the service.

A client can be considered trusted once it has been granted access to the service.

- Target is the service or destination that is protected by the MTD system.
- Sink is destination for untrusted clients which can be an unroutable destination or simply a honeypot for monitoring.

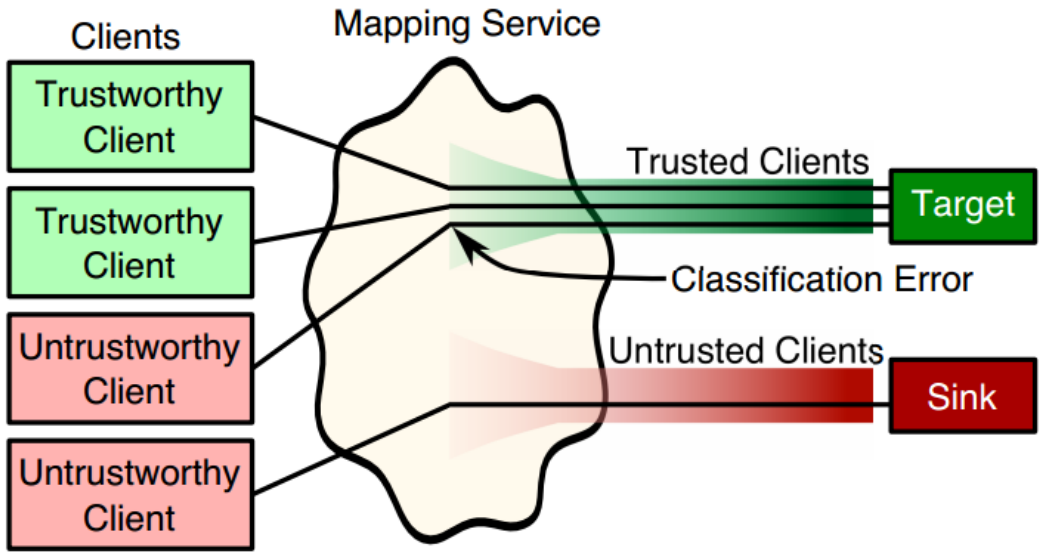

Figure 2.3: NMTD components

- Mapping System is the component that identifies and classifies clients as authorized or not.

Figure 2.3 from Green et. al illustrates the components of network MTDs and how they interact with each other.

In order to accomplish the goals of MTD, three main properties have been identified:

- Moving Property which forces clients to follow the mapping system component in order to reach the protected target and limits the attack surface of any untrusted clients. Three additional sub-properties have been identified in order to fulfill this property:
  - Unpredictability, which guarantees that the MTD system moves the targets in an unpredictable fashion in such a way that clients are unable to guess the location of any target unless they are authorized to do so.
  - Vastness, which guarantees that the movement space of the MTD system

is large enough such that it would be difficult for clients to exhaustively search the entire space.

  - Periodicity, which guarantees that the targets are moved regularly in such a way that any information gleaned by untrusted clients are rendered void quickly.

- Access Control Property requires that clients are only able to reach their target if and only if they are authorized by the mapping system. Three additional sub-properties have been identified to fulfill this property:

  - Uniqueness guarantees that every client is authorized individually and in such a way that authorization is restricted to that client.

  - Availability guarantees that each client authorized to a target will be able to successfully reach the target when required - the MTD technique must ensure that no DoS vulnerabilities are introduced.

  - Revocability ensures that the mapping system component is capable of termination or revoking any prior authorization to a client with affecting other clients or system components.

- Distinguishability Property ensures that the MTD system is capable of identifying trustworthy clients vs. untrustworthy clients. In order to do so, characteristics unique to trustworthy clients must be identified in order to determine which clients to authorize.

As observed from the techniques categorized above, Network-Based MTD techniques seem to be effective during the reconnaissance phase of attackers due to the nature of network configurations - IP Addresses and Port Numbers, in comparison to

Application-Based MTD approaches where dynamic randomization is possible in order to thwart ongoing attacks, for example introducing randomness to SQL queries in such a way that attackers are unable to correct guess the syntax of the query. In order to be truly effective, Network-Based MTD should be incorporated into the Defense-In-Depth approach of current Network Security practices. Furthermore, Network-Based MTD approaches are restricted in the 'moving' aspect of MTD due to the reliance on two components: Intrusion/Anomaly detection in order to identify anomalous or suspicious traffic and reliance on an authentication service in order to identify users or hosts that are authorized to access the service.

Intrusion detection and anomaly detection is a problematic approach due to the fact that it is restricted in scope. For instance, anomaly-based and signature-based detection systems are often prone to misclassification of data and high error rates. In addition, most successful attacks are sophisticated and complex in that they do not follow a predetermined pattern or that the malicious operator behind it modifies the attack in such a way that previously known patterns will fail. In the time it takes to update the information on the exploit, attackers would have already attempted and executed the exploit - making any response taken against attackers a *reactive* approach rather than MTD's fundamental concept of *proactive* approch. The use of authenticated users and hosts is also disadvantageous primarily due to its inflexible approach - although keeping track of each user and restricting access to those that have been authenticated reduces the probability of unauthorized access, scaling such MTD techniques to use larger networks, i.e. public internet access, is difficult due the large number of potential users. In addition, there is still the possibility of insider attacks or malicious users impersonating authenticated users.

Static and Dynamic MTD techniques In addition to the 5 classes of MTD techniques, each class can be further categorized. For instance, MTD mechanisms for programs can be categorized into two classes depending on if a program is running (*dynamic*) or not (*static*) at the time when moving happens. For instance, existing ASLR approaches are static, because the positions of code and data areas are only moved at the launch of a program but not when a program is running. On the other-hand, dynamic MTD techniques offer a wider option of choices for when-to-move, in exchange for being more difficult to implement due to other considerations: i.e. overhead cost and downtime during movement.

2.5 Key Challenges to Moving Target Defense

We have identified three key challenges in developing MTD systems, a brief explanation of each is provided below.

Ensure availability to legitimate users while disrupting attackers A big challenge in developing MTD systems is to ensure system availability to legitimate users while disrupting attackers. When changing system configurations or moving components, services to users should be undisturbed. Malicious users on the other hand, often attempt to use or access the system in unintended methods by exploiting vulnerabilities present. By changing the system configuration, legitimate users would still have unimpeded access to the service as they continue to use it in its intended way; on the other hand, malicious users attempts to exploit vulnerabilities would be disrupted as they rely on the previous system configuration.

System design must provide the intended security benefits (attack surface must reduced) In developing MTD systems, specific exploits are prevented

by changing certain components such that vulnerabilities leading to the exploits are unavailable. However, in doing so, developers must be careful that changing configurations does not introduce new vulnerabilities

Development of secure control system that handles Moving Target Defense system In complex MTD systems, the main logic handler needs to be secure. If MTD logic is compromised, the entire system is rendered null as malicious users will have knowledge of the new configuration as soon as it is available resulting in any configuration changes inefficient in increasing uncertainty for the attacker.

Scalability of MTD Technique As with most security techniques, MTD should be practical to use and therefore easy to scale with most systems.

2.6 Moving Target Defense Framework

Four components have been identified in defining MTD systems:

1. MTD Technique (What-to-Move), which identifies the component(s) to be modified in order to introduce complexity to attackers. These components can vary from identities within networks to physical devices.

2. MTD Approach (When-to-Move), which describes the logic used in deciding when and how often to modify the configuration. Various methods have been proposed ranging from pseudo-random to game-theoretic approaches to nature-inspired algorithms.

3. Applications of MTD, which identifies the area the MTD system is applicable to - hindering reconnaissance efforts or protect against unwanted modification and analysis.

4. Evaluation methods, which defines the methods used to determine whether the technique is 'effective' - researchers have proposed differing definitions of effectiveness such as successfully defending against a specific attack vs. hindering or disrupting reconnaissance efforts of an attacker.

Chapter 3

WEB APPLICATIONS

3.1 Web Applications: Individual and Commercial Front Doors

Web applications continue to remain as the most popular method for businesses to conduct services over the Internet. As the number of web applications that are accessible increase, so too does the amount of sensitive business and user data that is managed and processed by web applications. Because of their continuously increasing popularity and their inherent nature, vulnerabilities that are present in these web applications put both businesses and end-users' security and privacy at risk.

This is not an abstract risk, as the JPMorgan Chase breach in 2014 affected 76 million US households Silver-Greenberg *et al.* (2014). Bloomberg reported that the hackers "exploited an overlooked flaw in one of the bank's websites" Robertson and Riley (2014). Therefore, web applications serve as the "front door" for many companies and ensuring their security is of paramount importance.

Current techniques and tools focus primarily on prevention and discovery of these vulnerabilities. For instance, many techniques and tools using static analysis (white-box) or dynamic analysis (black-box) approaches have been proposed and developed to discover the vulnerabilities of web applications Balzarotti *et al.* (2008); Felmetsger *et al.* (2010); Jovanovic *et al.* (2010); Doupé *et al.* (2012, 2013), so that the vulnerabilities can be removed before attackers discover and exploit them. However, the efforts of discovering and fixing vulnerabilities are not enough to protect web applications for many reasons:

1. The increasing complexity of modern web applications brings inevitable risks

that cannot be fully mitigated in the process of web application development and deployment

2. Attackers are able to take their time in understanding the target web application's functionality and underlying technology stack before executing an attack.

We believe that a defense-in-depth approach is best in securing web applications. Therefore, to complement the aforementioned vulnerability analysis techniques, we propose to use the ideas of Moving Target Defense to create a novel and proactive approach that adds an additional layer of defense to web applications. At a high level, a Moving Target Defense dynamically configures and shifts systems over time to increase the uncertainty and complexity for attackers to perform probing and attacking Cui and Stolfo (2011); Zhuang *et al.* (2014). While a system's availability is preserved to legitimate users, the system components are changed in unpredictable ways to the attackers. Therefore, the attacker's window of attack opportunities decrease and the costs of attack increase. Even if an attacker succeeds in finding a vulnerability at one point, the vulnerability could be unavailable as the result of shifting the underlying system, which makes the environment more resilient against attacks.

To best apply the MTD ideas to protect web applications, there are two high-level decisions:

- Deciding what web application component to move
- Choosing the optimal frequency of randomization of the chosen components

To assist in answering these questions, we first dissect the architecture of a modern web application - both client and server as well as their running environments, in order to explore the possible application of MTD at different layers. We hope our

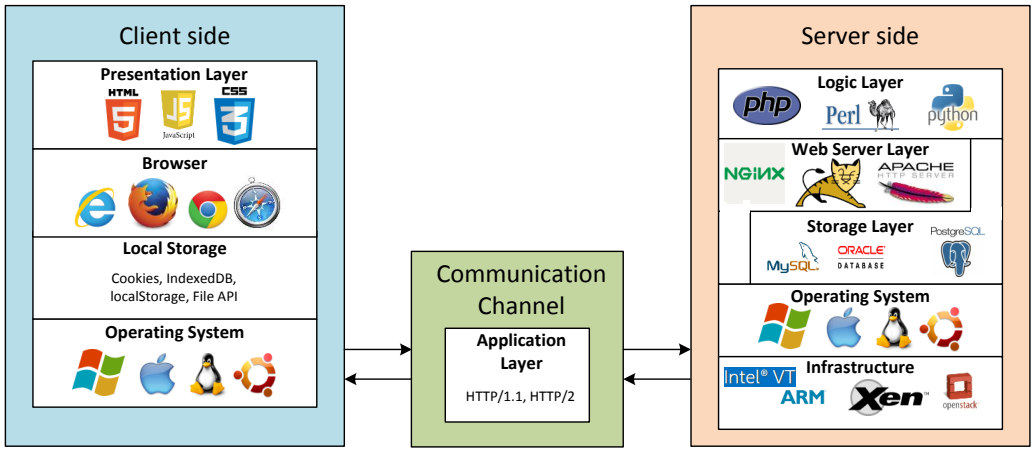

Figure 3.1: A Modern Web Application Architecture and Its Running Environments.

analysis provides insights into the trade-offs among the different places to apply MTD to web applications.

We also discuss our first steps in applying MTD techniques to protect web applications. The first technique changes the server-side language used in a web application by automatically translating server-side web application code to another language in order to prevent Code Injection exploits. The second technique shifts the database used in a web application by transforming the backend SQL database into different implementations that speak different dialects in order to prevent SQL Injection exploits.

3.2 Dissecting Modern Web Applications

In order to properly understand how to apply the ideas of moving target defense to web applications, we first describe a typical web application followed by a discussion on the ideas behind using moving target defense. As shown in Figure 3.1, a web application follows a distributed application structure, with components running on both server and the client systems. When requesting a web resource, the client first

issues a request to the server-side component over its communication channels - this is typically the HTTP protocol and its derivative protocols such as HTTPS, SPDY, and HTTP/2. The server receives the request and processes it using the application's logic and returns the requested resource. If data stored in an external database is requested, the server passes the relevant user-input as a query and processes the result. The server side typically includes the following layers from top to bottom [1] :

- The server-side logic layer implements the application business logic using high-level programming languages, such as Java, PHP, or Python.
- The web server layer receives the HTTP request from the client, parses the HTTP request, and passes the request to the appropriate server-side program. Examples include Apache web server, Windows IIS, or Nginx.
- The data storage layer stores the web application state and user data. Popular data storage systems are traditional SQL databases, which include MySQL, PostgreSQL, or MSSQL.
- The operating system layer that provides the running environment for the web server layer and database storage layer.
- The infrastructure layer that runs the operating systems. An infrastructure could be a physical machine or a virtualization platform which manages multiple virtual machines.

The client receives the HTTP response from the server-side component and converts the HTML contained in the HTTP response into a graphical interface for the user. The client consists of the following components:

[1] Of course, modern web application stacks can become increasingly complex, with caches, external requests, or other services, however we restrict our discussion to this abstracted model.

- The client-side logic layer, usually known as the presentation layer. The logic code here is usually composed of a combination of HTML, CSS, and JavaScript, with JavaScript providing a way for the server-side code to execute application logic on the client.

- The browser, which retrieves the presentation layer code from the server (typically HTML), interprets it, and presents it as a graphical interface to the user.

- The storage layer, that the presentation layer code uses to store data. Available storage methods include `cookies`, `localStorage`, `IndexedDB`, and File APIs.

- The operating system layer, which the browser runs on.

3.3 Current Security Issues and Considerations

Based on our proposed definition of a web application's structure in Figure 3.1, if a layer is compromised, the upper layers are not trustworthy. For instance, if the server's operating system is compromised, then the data storage, web server, and server-side logic are also compromised due to the interconnected nature of web applications. In addition to this, if the communication channel is also compromised - i.e. Man in the Middle attack; the client side presentation layer also gets affected, as attackers are able to manipulate the information being seen by users. In order to attack a layer in Figure 3.1, adversaries often utilize interfaces exposed to the upper layers. For instance, in a heap spraying attack executed on the client browser layer Ratanaworabhan *et al.* (2009), an attacker allocates malicious objects using JavaScript in the presentation layer in order to coerce the browser into spraying objects in the heap, increasing the success rate of an exploit where a vulnerability is exploited by jumping to the location within the heap. In this example, the attacker leverages a vulnerability located in the presentation layer - lack of input validation; to

exploit a vulnerability in the browser layer that leads to arbitrary code execution in the browser's address space. The arbitrary code that was injected can in turn exploit a vulnerability found on the client operating system in order to escalate a malicious user's privilege and further infect the client machine. Furthermore, vulnerabilities are not isolated within each system. For example, malicious JavaScript code could be delivered by an attacker by exploiting a vulnerability in the server-side logic layer, using a reflected or stored cross-site scripting (XSS) vulnerability.

Chapter 4

MOVING TARGET DEFENSE FOR WEB APPLICATIONS

4.1 Current State of Moving Target Defense in Web Applications

Current techniques and approaches to web vulnerabilities focus on detection, patching, and prevention. In addition to these traditional approaches, recent attempts have been made to apply the moving target defense concept to web applications. Huang *et al.* proposed to create and rotate between a set of virtual servers, each of which is configured with a unique software mix, to move the attack surface for web surfaces Huang and Ghosh (2011). Their work also explored the various opportunities of diversification in the web application software stack, providing a higher-level overview of the attack surface. Our work builds on this by further analyzing the components in each layer and defining what randomization in each layer entails; in addition to attempting to automate diversification of the components located in the logic and storage layer. Aiming to prevent SQL injection attacks, Boyd *et al.* proposed to create instances of unpredictable database query languages and to translate them to standard SQL using an intermediary proxy Boyd and Keromytis (2004). Although their approach also aims to prevent SQL injections, our proposed diversification approach aims to prevent a broader range of vulnerabilities—specifically unpatched vulnerabilities, zero day exploits, and mass-attacks targeting specific database implementations. Portner *et al.* proposed to defend against cross-site scripting (XSS) attacks by mutating the symbols in JavaScript in such a way that maliciously injected JavaScript code fails to execute due to incorrect version compatibility, and identifying such malicious programs Portner *et al.* (2014). Their work aims to prevent a

different class of vulnerabilities, specifically located at the presentation layer on the client side. On the other hand, our proposed approach is aimed at applying MTD ideas on the server side of the web application architecture - specifically the logic and storage layers. Despite these differences, we envision such techniques, located in each layer, to cooperate together to provide a defense-in-depth approach in defending web applications.

4.2 Leveraging web application layers for MTD

The core idea of moving target defense (MTD) can be applied to each layer of web applications and their running components. The key consideration to ensure however, is that the "movement," done successfully prevents the intended vulnerability or exploit, while preserving the application functionality. In this section, we discuss the different components that are available for moving at each layer of the web application. Specifically, we focus on the layers that play a major role in web applications and those that are often targeted: the logic layer, storage layer, and presentation layer, and browsers. For a discussion on layers that are common to other applications, which include the operating system layer and the infrastructure layer, we refer the interested reader to research in these areas Team (2003); Barrantes *et al.* (2003); Larsen *et al.* (2014); Wartell *et al.* (2012); Williams *et al.* (2009); Salamat *et al.* (2011); Vikram *et al.* (2013); Dunlop *et al.* (2011); Carvalho and Ford (2014); Li *et al.* (2014).

4.2.1 Logic Layer

At the logic layer of web applications, there exist at least two ways of applying MTD by changing it's implementation. The first approach is based on the idea of software diversity Larsen *et al.* (2014), changing and modifying the code at statement, function, or object levels. This technique is widely used in lower level lan-

guages, and is typically used to prevent memory corruption vulnerabilities - specifically return-oriented programming (ROP) exploits, that take advantage of previously known code-layouts by hijacking the program control flow and executing malicious code. This automated diversity MTD technique can be done statically or dynamically. Web applications however, are typically written in higher level languages such as Java, Python, and Ruby - offering them some degree of immunity to memory corruption vulnerabilities. As a result, most vulnerabilities found in web applications are a inherent in the code itself - for instance, XSS attacks, where server-side web application code is able to modify or create HTML from user input that is unsanitized and untrusted. In such a case, software diversity will be ineffective as the problem lies within the logic of the web application.

Another possible MTD approach at the logic layer is similar to software diversification, with the concept of extending 'diversification' to a higher level. By switching a web application's implementation from one language to another, one could eliminate some language- or framework-specific vulnerabilities, due to some vulnerability classes being specific to certain programming languages. For instance, an application that is developed with *Ruby on Rails 3.0.5* may introduce execution-after-redirect vulnerabilities, while its counterpart developed with *Python* and *Django 1.2.5* is impervious to this class of vulnerabilities, primarily due to the different implementations of the underlying framework Doupé *et al.* (2011). As with software diversity, changing the web application's implementation language could be static or dynamic. In a static *implementation language* switch, the translation of the original application is done prior to launch and allows the web server to simply launch another instance of the the application that is written in a different implementation language. To automate the process, web application developers need only develop the application once in their preferred language followed by feeding their original code to a translator program to

translate the code into functionally equivalent code in another web application language. The process of translation between languages is difficult due to several reasons. The primary issue being that the original language the code is written in might have some features that the target language does not offer. In a dynamic *implementation language* switch, in addition to the translation issues in the static approach, resources must be managed properly when switching - for example, executing the translation while managing ongoing requests and responses, as well as ensuring that the states of the running web application is maintained or transformed in order for the program to understand once translation is completed. In Section 5.1 we further discuss our implementation of this automated *implementation language* switch idea.

4.2.2 Storage Layer

The primary challenge that the storage layer of web applications face are often database injection attacks wherein, user data retrieved from the logic layer is interpreted as SQL statements by the application's database management systems. In order to successfully perform these SQL injection attacks, malicious users need to carefully craft their input to overcome injection prevention techniques by utilizing reversed tokens in the target SQL syntax and modify the logic layer's intended SQL statement.

While SQL itself is standard, there exist different SQL database implementations that utilize slightly different SQL syntaxes (also called dialects). By leveraging this information, one can switch the database implementation used in a web application in order to defeat targeted SQL injection exploits aimed at specific SQL dialects. For instance, when using a database that uses MySQL's dialect, both single (`''`) and double (`""`) quotations are used for identifying values—on the other-hand, switching a database that contains the same data written in PostgreSQL's dialect results in single

quotations being restricted for values and double quotations used when identifying field names, table names, etc.

Similar to the logic layer implementation randomization, static MTD for databases can be realized by exporting the data from one database implementation prior to execution and then importing it into a different database implementation. Dynamic MTD for the storage layer is also possible to achieve by translating the database at intervals and swapping the currently running instance with the target instance. However, both databases need to be kept synchronized while allowing for continuous external interaction. In Section 5.2 we discuss our implementation of this idea.

4.2.3 Presentation Layer

The client side presentation layer primarily contains technologies that are most directly accessible to the user - they provide information to the browser that allows users to view their requested resource. Some examples of technologies found in this layer are: client-side JavaScript code running some of the web application's logic, the HTML DOM containing the web page layout and provides a way for users to interact with the application through forms, radio buttons, and hyper-links, and CSS that manages the web page style and layout. At this layer, the most direct threat to user data is found through XSS attacks wherein malicious scripts are injected into the web application in order to steal information from users.

Several MTD approaches have been proposed to prevent against such attacks. One technique is to introduce a degree of randomness to the underlying HTML form fields by adding tags to each field in order to obfuscate or hide their real values against web bots looking to automate attacks Vikram *et al.* (2013). Another approach was proposed to introduce randomness to the JavaScript code by mutating tokens in such a way that malicious JavaScript code injected by attackers will fail to execute as

their input does not match the version running on the application. In addition to this random token, multiple versions of the website utilizing varying JavaScript versions are also deployed to further confuse attackers Portner *et al.* (2014).

4.2.4 Browsers

Most modern web browsers have modularized architectures that typically include rendering engines, JavaScript interpreters, and XML parsers Reis *et al.* (2009). By moving and modifying these components, vulnerabilities present within the components of the browser can also be mitigated. By doing so, the browser itself, as well as the underlying system it is running on can be defended. For example, the Cheetah browser and the 360 browser can change their rendering engines between WebKit and Trident. By doing so, remote code executions and memory corruption exploits can be prevented.

In addition to defending browsers against exploits, user privacy can also be protected by modifying browser configurations. Each browser instance has its own unique configurations: System fonts, Browser plugins and versions, cookie enabled flags, and user agents just to name a few. Using these information, web applications are able to uniquely fingerprint a browser in order to track users and their web movements Eckersley (2010). By diversifying the configuration components, a browser can be prevented from being fingerprinted, thereby protecting the web user's privacy Laperdrix *et al.* (2015); Nikiforakis *et al.* (2015, 2013).

Chapter 5

IMPLEMENTATION

5.1 Source Code language diversification

In order to apply the concept of MTD onto the server side logic layer, we aim to change the underlying language implementation of the web application while ensuring that the main functionalities and front facing interface remain unchanged. This is to keep the idea of misleading potential attackers by hiding underlying changes while keeping things consistent for legitimate users. We modify the source code in such a way to prevent certain categories of vulnerabilities from being effectively exploited — for instance, remote code injection exploits would become ineffective due to a) payloads developed by an attacker would rely on information before code-manipulation, potentially rendering them invalid; or b) malicious code that an attacker manages to inject into the application logic would be ineffective, as again, it would rely on previously known system information. In addition, because of the method of changing the underlying code, undiscovered or unpatched vulnerabilities (zero-day exploits) present in the original language of the web application would be protected against exploits.

We first describe our implementation of a static MTD mechanism for the logic layer. In order to simplify the translation process, we choose to narrow down to two languages to convert between. PHP and Python were selected as the base code languages after sampling different open source web applications and production-ready web applications, both widely used as web application code bases that include Google, YouTube, Pinterest, and Bing. In order achieve a MTD system applicable to web

applications, we automate the process of converting a Python web application to PHP and vice-versa. The first step to achieve this MTD system is to convert the original source code into a syntactically valid output in the target language. In the case of our chosen languages, to translate between PHP and Python requires further refinement of Python web applications as there exist varying web application frameworks for Python (Django, web2py, and Flask to name a few) while PHP remains primarily suited for web development and web application creation. We choose to focus on Python applications developed using `cg-lib` however, the translation and automation process can be extended to other frameworks in the future.

The functionality of our translator is similar to a compiler, as such, to translate from Python to PHP, we chose to leverage existing functionality as the initial step—specifically, we utilize the built-in `parsing` and `unparsing` modules in Python 2.7 to construct an Abstract Syntax Tree (AST) of the input Python program. The unparsing module creates the leaves and parsing module gets it back together. During this process, we were able to manipulate each leaf according to the destination language's grammar requirement. After feeding the web application's code into this parser, it creates a tree of objects representing the grammar. This tree is then sent to the `unparsing` module to be reversed back in to code. This module reads the tree and recursively calls print code to output the original code back again. The unparsing module is shown in Figure 5.1 and illustrates the a function call in Python with modulus operator. Once the AST of the original code is generated and translated, we convert the AST back to readable Python code using Python's `unparse` module - we develop a new library based on this module to generate PHP code instead of Python code. It is important to note that we did not modify the original intent of these modules - to take a full parse tree and print back the equivalent source code; instead, we simply modify the output of printing back readable source code. The core

```
def _Print(self, t):
self.fill("print ")
do_comma = False
if t.dest:
self.write(">>")
self.dispatch(t.dest)
do_comma = True
for e in t.values:
if do_comma:self.write(", ")
else:do_comma=True
self.dispatch(e)
if not t.nl:
self.write(",")
```

Listing 5.1: Original _Print in unparse to generate Python code.

idea of our implementation is that the parser creates the AST and sends it to the `unparsing` module and the unparsing module prints out the translated code instead of the original code that was fed.

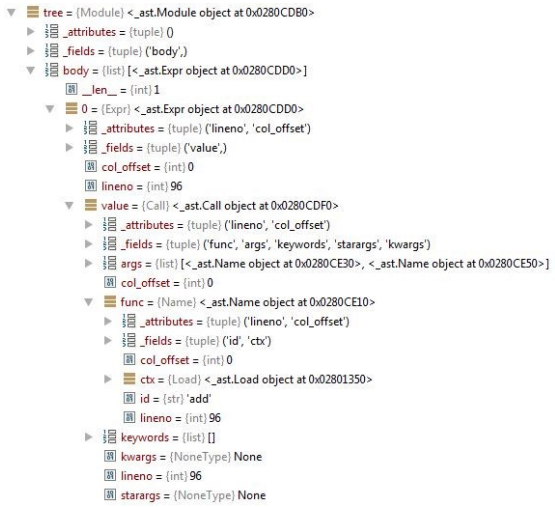

Figure 5.1: The Unparser

For instance, when translating a simple `print` statement in Python to the PHP equivalent of `echo`, we modify the `_Print` function in the `unparse` module as shown in Listing 5.1.

```
def _Print(self, t):
self.fill("echo ")
do_comma = False
if t.dest:
self.write(">>")
self.dispatch(t.dest)
do_comma = True
for e in t.values:
if do_comma:self.write(", ")
else:do_comma=True
self.dispatch(e)
if not t.nl:
self.write(",")
self.write(";")
```

Listing 5.2: Modified _Print in unparse to generate PHP code.

For this example, we replace the `print` Python keyword with the `echo` PHP keyword and ensure that the instruction is terminated with a semicolon as shown in Listing 5.2.

Once this step is completed, we have web application code that is in a syntactically valid PHP form. However, it does not have the same semantics as the original Python program nor is a semantically valid PHP application. As such, it needs to be further modified in order to satisfy these two conditions. This modification is necessary as there may not be a one-to-one translation of every feature from Python to PHP or vice-versa. To illustrate this problem, suppose we have a Python instruction to terminate and exit the program using the following code snippet:

```
sys.exit(0)
```

Once the initial syntactic translation is completed and a valid PHP output is generated, we have the following:

```
sys->exit(0);
```

This is now valid PHP code as it uses the correct notation for function calls and is

```
<?php
class sys{
public static function exit_Wrapper_PHP($command){
exit($command);
}
}
?>
```

Listing 5.3: sys class in PHP

correctly terminated with a semicolon. However, when executed, PHP sees this line of code as a call to some `sys` object with an call to an internal function `exit(0)`, which does not exist in PHP by default. The Python `sys.exit(0)` function call however does have a direct equivalent in PHP—the `exit($status)` function. We therefore leverage these direct equivalent function calls by implementing them as code shims in order to match the new function calls. To this end, because the syntax of the code is known to us after the initial translation is completed, we develop a PHP library that contains an object called `sys` with a function call to `exit(status)` as shown in Listing 5.3. This PHP library `shim` can be included in the translated application in order for the function call to remain semantically valid.

Shims Most, if not all programming languages include libraries that contain predefined functions that are called whenever a specific function call is used. When translated using our method, the target language will often have different libraries and corresponding functions to the original language; causing a mismatch in functionality in the syntactically matched output. Furthermore, developers have the capability to create and use custom libraries in addition to the standard, predefined ones - some overwriting/overloading the default functions. In such cases, these custom functions must be investigated thoroughly and the translator adjusted in order to ensure similar functionality to the original application intent.

One solution to this problem is to create shims for each of these predefined or custom functions. As described above, shims are simply custom-built libraries that have the ability to behave as if it is the library that was originally called by the application without affecting the flow of the application. For our implementation, we have created several PHP and Python shims that emulate built-in and developer-specified functions and libraries in the target language. Some samples of shims that we have implemented are: a CGI shim that allows the ability to emulate the CGI module from Python in PHP, a Time shim which allows the use of Python `localtime()` function in PHP and return an array of time variables in the order of Python's `localtime()`.

The creation of these shims can be extended to include other popular libraries; for instance, MySQLdb is a Python specified library for MySQL functionality that is required to be imported within the application code. Once the application is run through he syntactic converter, this library needs to call predefined PHP MySQL functions as a shim in order to ensure similar functionality to the original Python application. Some example MySQLdb functions that were translated and shimmed are `connect`, used to connect to the database and `rowcount`, used to display number of affected rows.

Context-Sensitive Approach As mentioned previously, simply translating between languages is not enough due to differing syntax and semantics. Furthermore, data structure differences between PHP and Python require specific handling of code based on its context - so called `context- sensitive translations`. One such instance is the use of the `%` operand in Python in order to perform string formatting. This operand can take different types of variables as input; once converted however, there is no direct counterpart in PHP. In order to handle the multi-variable ability of

the Python operation, the tool should be able to detect if such an operation is called and modify the Python code to output PHP code that handles the specific operation in that line.

Using this approach, we recursively run the tool on the Python functions that the original program calls, and convert them as well. If, for instance, the original program is written in C, we must create the equivalent function for that code in the target language.

5.2 Database dialect diversification

To enable the MTD concept in the server side storage layer, we propose to use the same method done in the logic layer by modifying the underlying database dialect of the web application. An integral part of this methodology is to ensure that at any time during this movement mechanism, the integrity and availability of data is preserved. Using this randomization technique, we anticipate that certain vulnerability and exploit categories will be rendered ineffective—specifically, we seek to protect against SQL injection exploits by leveraging the syntactical differences between various SQL dialect. By translating between different database dialects, the system could be potentially protected against database-specific exploits as well as unpatched or undiscovered vulnerabilities by rendering the information an attacker has about the system invalid due to differing web application configurations. We have identified two considerations to be made when performing the database translations:

- No alterations must be made to the data content —that is, at the end of the translation process, end-users must be able to see the same information stored regardless of the underlying database dialect.

- Access to the database and the data stored within must be available anytime

the application logic issues a request while keeping the translation process transparent to the user.

These two properties ensure that the web application service is undisturbed and the MTD mechanism is hidden to malicious users.

Similar to our source language translation methodology, we refine the selections of database dialects to two widely popular database dialects, MySQL and PostgreSQL. These two were chosen due to their high popularity rankings and the large enough differences in syntax - anticipating them to be the target of wide-spread database exploits. In addition, both dialects are used by well known entities as their back-end - Facebook, Google, Amazon, U.S. Dept of labor, U.S. State Department, and Sun Microsystemsto name a few.

As previously mentioned, we want to leverage the syntactical differences between SQL dialects in our MTD mechanism. In particular, some of the differences between MySQL and PostgreSQL syntaxes that we are interested in leveraging include the following:

- When starting comments in MySQL, the `#` or `--` (A space after the `--` is required) is utilized. On the other hand, to begin a comment in PostgreSQL, the `--` (the space is not required) is used.

- To identify values in MySQL, the single (`''`) quotes or double (`""`) quotes are used. When using PostgreSQL, single (`''`) quotes are used in identifying values while double (`""`) quotes are reserved in identifying field names, table names, etc.

- When doing string comparisons in MySQL, case-sensitivity is not taken into consideration. However, when string comparison is done in PostgreSQL, they are done in a case-sensitive manner, i.e. `john != JOHN != John`.

Each one of these syntax differences affect the SQL injection payloads written in order to take advantage of a web application's SQL injection vulnerability. If an attacker assumes that the web application is using a particular database backend; for instance, if the attacker scans the entire web for web applications that utilize a specific version of MySQL that has a injection exploit using their value identification method, changing the dialect to PostgreSQL will cause the MySQL payload to fail.

Similar to the source code language diversification approach, we choose to develop a tool that can automate the conversion or translation between database dialects. In order to convert from PostgreSQL to MySQL, we modified an existing open-source tool created by Lightbox that originally functions to convert PostgreSQL to MySQL—although we are able to simply create a database dump from PostgreSQL, using the resulting code is insufficient as there are still differences between the database syntaxes and data types that must be resolved. In order to acquire the initial database dump, specific run-time options must be enabled when dumping the database. PostgreSQL database dumps need to created while enabling insert statements by using the `--inserts` flag enabled to properly include the data stored while MySQL needs to have the `--compatible=postgresql` flag to properly include PostgreSQL keywords in the output file. To resolve the syntax mismatch during translation, we modify the code to process the original database dump by parsing through the input file (the source database dump) and replacing any PostgreSQL keywords and data-types into corresponding MySQL keywords and data types. In translating between different database data types, some considerations need to be made. For instance PostgreSQL's `BYTEA` can be equivalent to any of the MySQL data types shown in Table 5.1. When choosing what data type to convert to, it needs to be generic enough in such a way that it covers the possible data value found in the original database, while attempting to be as performant as possible. To handle conversion in the reverse direction, from

MySQL	PostgreSQL
BINARY(n)	BYTEA
VARBINARY(n)	BYTEA
TINYBLOB	BYTEA
BLOB	BYTEA
MEDIUMBLOB	BYTEA
LONGBLOB	BYTEA

Table 5.1: Comparison of MySQL and PostgreSQL Data Types.

MySQL to PostgreSQL, we simply re-purposed the code by reversing the process—that is, we parse through the dump file looking for MySQL keywords and data-types converting them to the corresponding PostgreSQL keywords and data-types. Similar to translating the language constructs of a web application, database translations may be costly as well depending on the size and complexity of the database—for instance optimizations done on the original implementation of the database will become invalid once converted.

Chapter 6

EVALUATION AND RESULTS

6.1 Application Functionality

In order to ensure system availability is preserved through the translation process, we compare the application functions to ensure that the same input returns the same output before and after the translation. In order to accurately quantify the results, we utilize Selenium to automate to the testing process. First, manual navigation of the web application is done in order to keep track of the main application functionality. In addition, this step also keeps track of the actions and input a regular user must take in order to complete each function as well as identifying the input form field names that is used on each page in the test automation process. Once a test suite is completed for the original application functions, the MTD technique proceeds to translate it into other target language - at this point, the underlying language has changed but the front-end of the application remains unchanged to users. The selenium test suite is run on this translated application using the same input as before; application functionality is preserved if the resulting output is similar to the output before translation. The MTD technique should be transparent to legitimate users, meaning the output should be the same to them no matter what. The results for the basic functionality of the collab application is show in Figure 6.1. Pre-translation output is done using the original configuration of the web application that uses Python and MySQL as the backend while post-translation output is shown using PHP and MySQL.

Results show that translating the web application leaves the interface unchanged to the user. Functionality is kept consistent through the use of code shims as illus-

	Note collaboration web application		
	Form Field and input	Pre-translation output	Post-translation output
Upload Notes			
	Date Created: August 09, 2017	Upload Succesful! popup, class note appears in "Show all classes" page	Upload Succesful! popup, class note appears in "Show all classes" page
	Class Name: Test class		
	Class Number: CSE210		
	Semester: Fall		
	Year: 2017		
	Professor: Test professor		
	File selected: SampleNotes.txt		
Show All Classes			
	Select "Show All Classes"	Web page updates to display a table of classes with notes uploaded	Web page updates to display a table of classes with notes uploaded
Show notes for one class			
	From "Show all classes" page, select a specific class	Web page updates to display uploaded notes for the specified class. Each note has a quality indicator a long with input to increase or decrease quality.	Web page updates to display uploaded notes for the specified class. Each note has a quality indicator a long with input to increase or decrease quality.
		Rank note with good quality	Rank note with good quality
		Rank note with bad quality	Rank note with bad quality
Rank class note with good quality			
	Select "Good Quality" button	Quality counter incremented by 1	Quality counter incremented by 1
Rank class note with bad quality			
	Select "Bad Quality" button	Quality counter decremented by 1	Quality counter decremented by 1

Figure 6.1: Application functionality results of collab web application

trated by having the same output before and after changing the underlying language of the web application using the same input.

6.2 Translation and Randomization cost overhead

To measure translation time overhead, a simple timer is used in the script when a configuration change starts and ends. For example, when a database change is invoked a timer starts in the script prior to calling the database translator and stops when the translation is done. This isolates the translation of each configuration change during each 'movement'. The measurements were taken using virtual machines from Oracle VirtualBox 5.1.18, four instances each running a unique configuration of the web application source language and database dialect on Ubuntu 12.04.5 with 1GB of memory. The combination for each system is as follows:

- 192.168.1.101 : PHP/MySQL

- 192.168.1.102 : PHP/PostgreSQL
- 192.168.1.103 : Python/MySQL
- 192.168.1.104 : Python/PostgreSQL

In addition, a fifth virtual machine was also set up to handle the request routing to the appropriate application configuration as well as handle the database of the application.

Figure 6.2 shows the average time it takes to completely change configuration from one variant to another. Similar colored cells signify that the configuration change is identical with the following colors corresponding to the configuration change - pink: same source language, differing database dialect, orange: differing source language, same database dialect, and green: differing source language and differing database dialect. Three data-sets were collected that differ in how long each test were run.

	2 Hour dataset	12 hour dataset	12 hour dataset
PHP/MySQL to PHP/PostgreSQL	599.6111111	503.6746988	647.2788462
PHP/MySQL to Python/MySQL	70.4375	67.18269231	98.81111111
PHP/MySQL to Python/PostgreSQL	1408.909091	420.3717949	572.3434343
PHP/PostgreSQL to PHP/MySQL	399.4	594.9764706	642.990566
PHP/PostgreSQL to Python/MySQL	390.8125	468.195122	920.0581395
PHP/PostgreSQL to Python/PostgreSQL	69.72222222	61.37634409	95.73626374
Python/MySQL to PHP/MYSQL	71.3125	66.78571429	113.0588235
Python/MySQL to PHP/PostgreSQL	371.6	434.9777778	578.4271845
Python/MySQL to Python/PostgreSQL	385.3076923	687.4123711	584.258427
Python/PostgreSQL to PHP/MySQL	413.7692308	893.2098765	749.5980392
Python/PostgreSQL to PHP/PostgreSQL	71.05882353	66.01149425	95.73684211
Python/PostgreSQL to Python/MySQL	506.8333333	474.89	659.2178218

Figure 6.2: Average time of changing configurations

Figure6.3 shows the various translation times between each MTD variants among all three datasets: database-only variants, source-code only variants, and database and source code variants.

Several data points appear further from the average translation times due to hardware issues while running the experiment, the virtual machines were running on

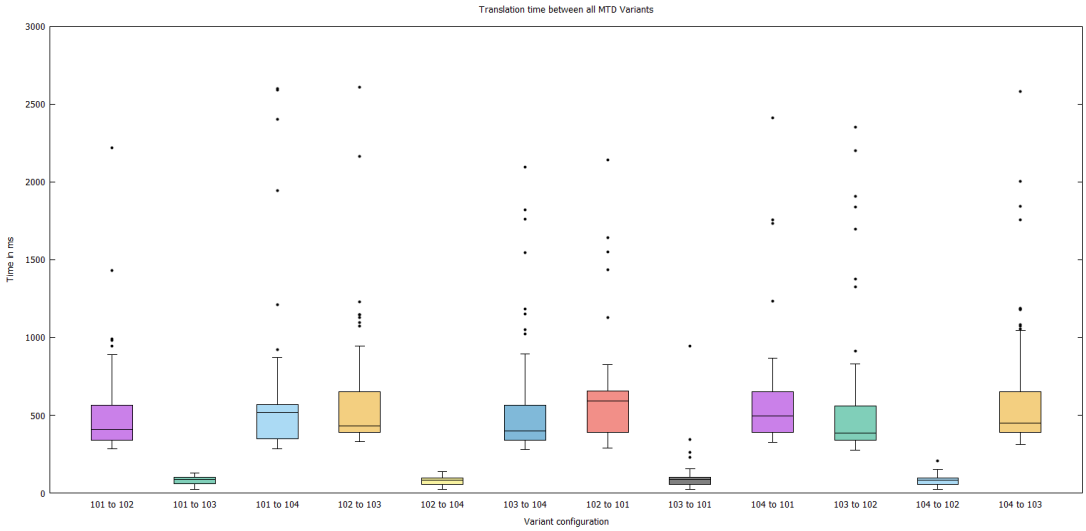

Figure 6.3: Translation time between all MTD variants

an unused laptop running an i7-4710 and 8GB of RAM. A closer view of the data set of one configuration translated to the other two is found in Figure6.4

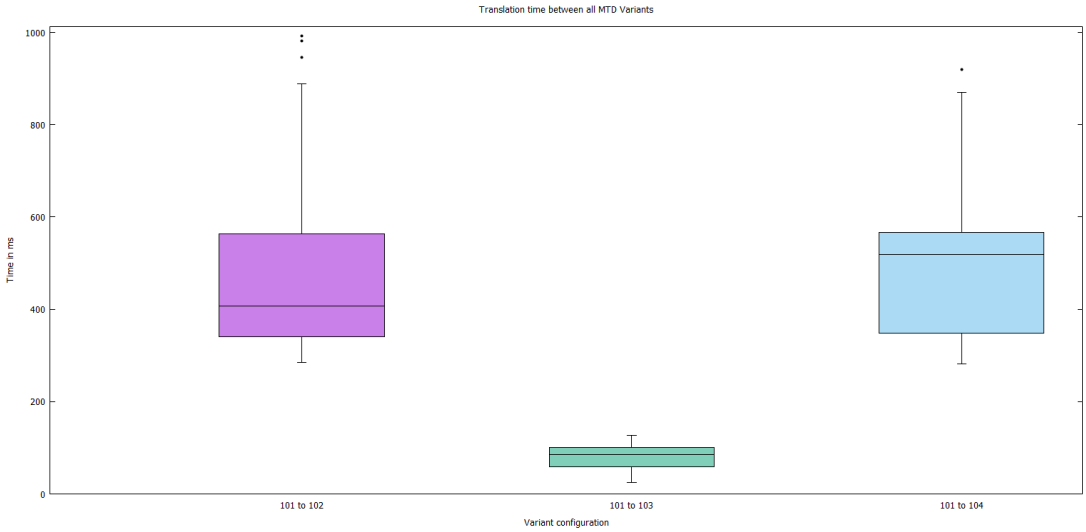

Figure 6.4: Translation from one configuration to two other configurations (Same language, different database dialect and different language, same database dialect)

From the data collected, it would seem that changing database configurations take less time than changing source language configurations. This can be attributed to the complexity in rebuilding the source-code to match the target language - most

languages are inherently different in code structure and semantics, requiring line by line translation to achieve the correct output. Database translation on the other hand is simpler as our chosen dialects both follow the ANSI/ISO SQL standard - the largest difference being differences in syntax rather than semantics.

6.3 Security Evaluation

To evaluate the security benefits of our MTD technique, we simply run the original application using a malicious payload to verify that the vulnerability is present and run the same payload after changing the configuration to verify that the MTD technique renders the payload invalid.

Chapter 7

CONCLUSION

In this thesis, I explored the feasibility of applying MTD concepts to web applications in order to introduce another layer of defense. A study was done on existing MTD techniques to identify the components and considerations in MTD. Using this knowledge, an analysis of the modern web application stack was done in order to understand how and where MTD can be applied. I developed an automated tool that assists in translating between source-code languages and database dialects in order to mitigate several classes of web application vulnerabilities. Current results show that employing the proposed MTD technique can defend against specified exploits when an exploit is launched enmasse against a specific technology (ex. mass attacks against postgres database web applications). When used to defend against targeted attacks, the MTD technique may not be as effective, as malicious users can eventually discover the simplistic configuration change pattern.However, implementing the source or database randomization with other MTD techniques such as network randomization can introduce another layer of complexity that an attacker must go through. The automated tool introduced can be extended to apply to any source language - however, the number of shims that need to be developed in order to have semantically equivalent web application may be take considerable amount of time as each application could potentially contain multiple custom functions. Providing a central repository for these code shims may reduce the amount of code repetition needed, this solution is infeasible however as having a public collection of these shims also enables easy access to malicious users. Further work is needed to develop a more generalized method of translating between languages in a more effective manner.

REFERENCES

Ammann, P. E. and J. C. Knight, "Data diversity: An approach to software fault tolerance", Computers, IEEE Transactions on **37**, 4, 418–425 (1988).

Antonatos, S., P. Akritidis, E. P. Markatos and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization", in "Proceedings of the 2005 ACM Workshop on Rapid Malcode", WORM '05, pp. 30–40 (ACM, New York, NY, USA, 2005), URL `http://doi.acm.org/10.1145/1103626.1103633`.

Avizienis, A. and L. Chen, "On the implementation of n-version programming for software fault tolerance during execution", in "Proc. IEEE COMPSAC", vol. 77, pp. 149–155 (1977).

Balzarotti, D., M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna, "Saner: Composing static and dynamic analysis to validate sanitization in web applications", in "Security and Privacy, 2008. SP 2008. IEEE Symposium on", pp. 387–401 (IEEE, 2008).

Barrantes, E. G., D. H. Ackley, T. S. Palmer, D. Stefanovic and D. D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks", in "Proceedings of the 10th ACM conference on Computer and communications security", pp. 281–289 (ACM, 2003).

Boyd, S. W. and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks", in "Applied Cryptography and Network Security", pp. 292–302 (Springer, 2004).

Carvalho, M. and R. Ford, "Moving-target defenses for computer networks", Security Privacy, IEEE **12**, 2, 73–76 (2014).

Crosby, S., M. Carvalho and D. Kidwell, "A layered approach to understanding network dependencies on moving target defense mechanisms", in "Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop", CSIIRW '13, pp. 36:1–36:4 (ACM, New York, NY, USA, 2013), URL `http://doi.acm.org/10.1145/2459976.2460017`.

Cui, A. and S. J. Stolfo, "Symbiotes and defensive mutualism: Moving target defense", in "Moving Target Defense", pp. 99–108 (Springer, 2011).

Cyberspace, T., "Strategic plan for the federal cybersecurity research and development program", Executive Office of the President National Science and Technology Council (2011).

Doupé, A., B. Boe, C. Kruegel and G. Vigna, "Fear the ear: discovering and mitigating execution after redirect vulnerabilities", in "Proceedings of the 18th ACM conference on Computer and communications security", pp. 251–262 (ACM, 2011).

Doupé, A., L. Cavedon, C. Kruegel and G. Vigna, "Enemy of the State: A State-Aware Black-Box Vulnerability Scanner", in "Proceedings of the USENIX Security Symposium (USENIX)", (Bellevue, WA, 2012).

Doupé, A., W. Cui, M. H. Jakubowski, M. Peinado, C. Kruegel and G. Vigna, "deDacota: Toward Preventing Server-Side XSS via Automatic Code and Data Separation", in "Proceedings of the ACM Conference on Computer and Communications Security (CCS)", (Berlin, Germany, 2013).

Dunlop, M., S. Groat, W. Urbanski, R. Marchany and J. Tront, "Mt6d: A moving target ipv6 defense", in "MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011", pp. 1321–1326 (2011).

Eckersley, P., "How unique is your web browser?", in "Privacy Enhancing Technologies", pp. 1–18 (Springer, 2010).

Felmetsger, V., L. Cavedon, C. Kruegel and G. Vigna, "Toward automated detection of logic vulnerabilities in web applications", in "USENIX Security Symposium", pp. 143–160 (2010).

Forrest, S., A. Somayaji and D. H. Ackley, "Building diverse computer systems", in "Operating Systems, 1997., The Sixth Workshop on Hot Topics in", pp. 67–72 (IEEE, 1997).

Ge, L., W. Yu, D. Shen, G. Chen, K. Pham, E. Blasch and C. Lu, "Toward effectiveness and agility of network security situational awareness using moving target defense (mtd)", vol. 9085, pp. 90850Q–90850Q–9 (2014), URL `http://dx.doi.org/10.1117/12.2050782`.

Green, M., D. C. MacFarland, D. R. Smestad and C. A. Shue, "Characterizing network-based moving target defenses", in "Proceedings of the Second ACM Workshop on Moving Target Defense", pp. 31–35 (ACM, 2015).

Huang, Y. and A. K. Ghosh, "Introducing diversity and uncertainty to create moving attack surfaces for web services", in "Moving Target Defense", pp. 131–151 (Springer, 2011).

Jafarian, J. H., E. Al-Shaer and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking", in "Proceedings of the First Workshop on Hot Topics in Software Defined Networks", HotSDN '12, pp. 127–132 (ACM, New York, NY, USA, 2012), URL `http://doi.acm.org/10.1145/2342441.2342467`.

Jafarian, J. H. H., E. Al-Shaer and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers", in "Proceedings of the First ACM Workshop on Moving Target Defense", MTD '14, pp. 69–78 (ACM, New York, NY, USA, 2014), URL `http://doi.acm.org/10.1145/2663474.2663483`.

Jovanovic, N., C. Kruegel and E. Kirda, "Static analysis for detecting taint-style vulnerabilities in web applications", Journal of Computer Security **18**, 5, 861–907 (2010).

Kewley, D., R. Fink, J. Lowry and M. Dean, "Dynamic approaches to thwart adversary intelligence gathering", in "DARPA Information Survivability Conference amp; Exposition II, 2001. DISCEX '01. Proceedings", vol. 1, pp. 176–185 vol.1 (2001).

Laperdrix, P., W. Rudametkin and B. Baudry, "Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification", in "Proceedings of the International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'15)", (2015).

Larsen, P., A. Homescu, S. Brunthaler and M. Franz, "Sok: Automated software diversity", in "Security and Privacy (SP), 2014 IEEE Symposium on", pp. 276–291 (IEEE, 2014).

Li, Y., R. Dai and J. Zhang, "Morphing communications of cyber-physical systems towards moving-target defense", in "Communications (ICC), 2014 IEEE International Conference on", pp. 592–598 (2014).

Luo, Y.-B., B.-S. Wang and G.-L. Cai, "Effectiveness of port hopping as a moving target defense", in "Security Technology (SecTech), 2014 7th International Conference on", pp. 7–10 (2014).

Luo, Y.-B., B.-S. Wang and G.-L. Cai, "Analysis of port hopping for proactive cyber defense", International Journal of Security and Its Applications **9**, 2, 123–134 (2015).

Michalski, J., C. Price, E. Stanton, E. Lee, K. Chua, Y. Wong and C. Tan, "Network security mechanisms utilizing dynamic network address translation", (2002).

Nguyen-Tuong, A., D. Evans, J. C. Knight, B. Cox and J. W. Davidson, "Security through redundant data diversity", in "Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on", pp. 187–196 (IEEE, 2008).

Nikiforakis, N., W. Joosen and B. Livshits, "PriVaricator: Deceiving fingerprinters with Little White Lies", in "Proceedings of the International World Wide Web Conference (WWW)", (2015).

Nikiforakis, N., A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens and G. Vigna, "Cookieless Monster: Exploring the Ecosystem of Web-based Device Fingerprinting", in "Proceedings of the IEEE Symposium on Security and Privacy", (2013).

Okhravi, H., M. Rabe, T. Mayberry, W. Leonard, T. Hobson, D. Bigelow and W. Streilein, "Survey of cyber moving target techniques", Tech. rep., DTIC Document (2013).

Pettis, K. and R. C. Hansen, "Profile guided code positioning", in "ACM SIGPLAN Notices", vol. 25, pp. 16–27 (ACM, 1990).

Portner, J., J. Kerr and B. Chu, "Moving target defense against cross-site scripting attacks (position paper)", in "Foundations and Practice of Security", pp. 85–91 (Springer, 2014).

Ratanaworabhan, P., V. B. Livshits and B. G. Zorn, "Nozzle: A defense against heap-spraying code injection attacks.", in "USENIX Security Symposium", pp. 169–186 (2009).

Reis, C., A. Barth and C. Pizano, "Browser security: lessons from google chrome", Queue **7**, 5, 3 (2009).

Robertson, J. and M. Riley, "JPMorgan Hack Said to Span Months Via Multiple Flaws", (2014).

Salamat, B., T. Jackson, G. Wagner, C. Wimmer and M. Franz, "Runtime defense against code injection attacks using replicated execution", Dependable and Secure Computing, IEEE Transactions on **8**, 4, 588–601 (2011).

Silver-Greenberg, J., M. Goldstein and N. Perlroth, "JPMorgan Chase Hacking Affects 76 Million Households", The New York Times (2014).

Team, P., "Address space layout randomization", Phrack (2003).

Vikram, S., C. Yang and G. Gu, "Nomad: Towards non-intrusive moving-target defense against web bots", in "Communications and Network Security (CNS), 2013 IEEE Conference on", pp. 55–63 (IEEE, 2013).

Wartell, R., V. Mohan, K. W. Hamlen and Z. Lin, "Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code", in "Proceedings of the 2012 ACM conference on Computer and communications security", pp. 157–168 (ACM, 2012).

Williams, D., W. Hu, J. W. Davidson, J. D. Hiser, J. C. Knight and A. Nguyen-Tuong, "Security through diversity: Leveraging virtual machine technology", Security & Privacy, IEEE **7**, 1, 26–33 (2009).

Zhuang, R., S. A. DeLoach and X. Ou, "Towards a theory of moving target defense", in "Proceedings of the First ACM Workshop on Moving Target Defense", MTD '14, pp. 31–40 (ACM, New York, NY, USA, 2014), URL `http://doi.acm.org/10.1145/2663474.2663479`.

Zhuang, R., S. Zhang, A. Bardas, S. DeLoach, X. Ou and A. Singhal, "Investigating the application of moving target defenses to network security", in "Resilient Control Systems (ISRCS), 2013 6th International Symposium on", pp. 162–169 (2013).

APPENDIX A

RAW DATA

```
--
-- PostgreSQL database dump
--

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;

DROP DATABASE test_database;
--
-- Name: test_database; Type: DATABASE; Schema: -; Owner: postgres
--

CREATE DATABASE test_database WITH TEMPLATE = template0 ENCODING = 'UTF8' LC_COLLATE
    = 'English_United_States.1252' LC_CTYPE = 'English_United_States.1252';


ALTER DATABASE test_database OWNER TO postgres;

\connect test_database

SET statement_timeout = 0;
SET lock_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SET check_function_bodies = false;
SET client_min_messages = warning;

--
-- Name: public; Type: SCHEMA; Schema: -; Owner: postgres
--

CREATE SCHEMA public;


ALTER SCHEMA public OWNER TO postgres;

--
-- Name: SCHEMA public; Type: COMMENT; Schema: -; Owner: postgres
--

COMMENT ON SCHEMA public IS 'standard_public_schema';


--
-- Name: plpgsql; Type: EXTENSION; Schema: -; Owner:
--

CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;


--
-- Name: EXTENSION plpgsql; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL_procedural_language';


SET search_path = public, pg_catalog;

SET default_tablespace = '';

SET default_with_oids = false;

--
-- Name: test_table; Type: TABLE; Schema: public; Owner: postgres; Tablespace:
--
```

```
CREATE TABLE test_table (
username character varying(15) NOT NULL,
password character varying(15) NOT NULL,
creditnumber numeric(10,0)
);


ALTER TABLE test_table OWNER TO postgres;

--
-- Data for Name: test_table; Type: TABLE DATA; Schema: public; Owner: postgres
--

INSERT INTO test_table VALUES ('user1', 'testpass', 60547381);
INSERT INTO test_table VALUES ('user2', 'pass', 72619472);
INSERT INTO test_table VALUES ('user3', 'passpass', 81124561);
INSERT INTO test_table VALUES ('user4', 'passwordpass', 5522698);


--
-- Name: username; Type: CONSTRAINT; Schema: public; Owner: postgres; Tablespace:
--

ALTER TABLE ONLY test_table
ADD CONSTRAINT username PRIMARY KEY (username);


--
-- Name: public; Type: ACL; Schema: -; Owner: postgres
--

REVOKE ALL ON SCHEMA public FROM PUBLIC;
REVOKE ALL ON SCHEMA public FROM postgres;
GRANT ALL ON SCHEMA public TO postgres;
GRANT ALL ON SCHEMA public TO PUBLIC;


--
-- PostgreSQL database dump complete
--
```

Listing A.1: Pre-conversion PostgreSQL Database Dump

```
# Converted with mysql2pg-1.9
# Converted on Wed, 01 Jul 2015 23:58:23 +0000
# Lightbox Technologies Inc. http://www.lightbox.ca

SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET time_zone="+00:00";

DROP DATABASE test_database;
CREATE DATABASE `test_database` DEFAULT CHARACTER SET UTF8;

USE `test_database`;

CREATE TABLE test_table (
username varchar(15) NOT NULL,
password varchar(15) NOT NULL,
creditnumber numeric(10,0)
) ENGINE=MyISAM;

INSERT INTO test_table VALUES ('user1', 'testpass', 60547381);
INSERT INTO test_table VALUES ('user2', 'pass', 72619472);
INSERT INTO test_table VALUES ('user3', 'passpass', 81124561);
INSERT INTO test_table VALUES ('user4', 'passwordpass', 5522698);
ALTER TABLE test_table
ADD CONSTRAINT username PRIMARY KEY (username);
```

Listing A.2: Post-conversion MySQL Database Dump

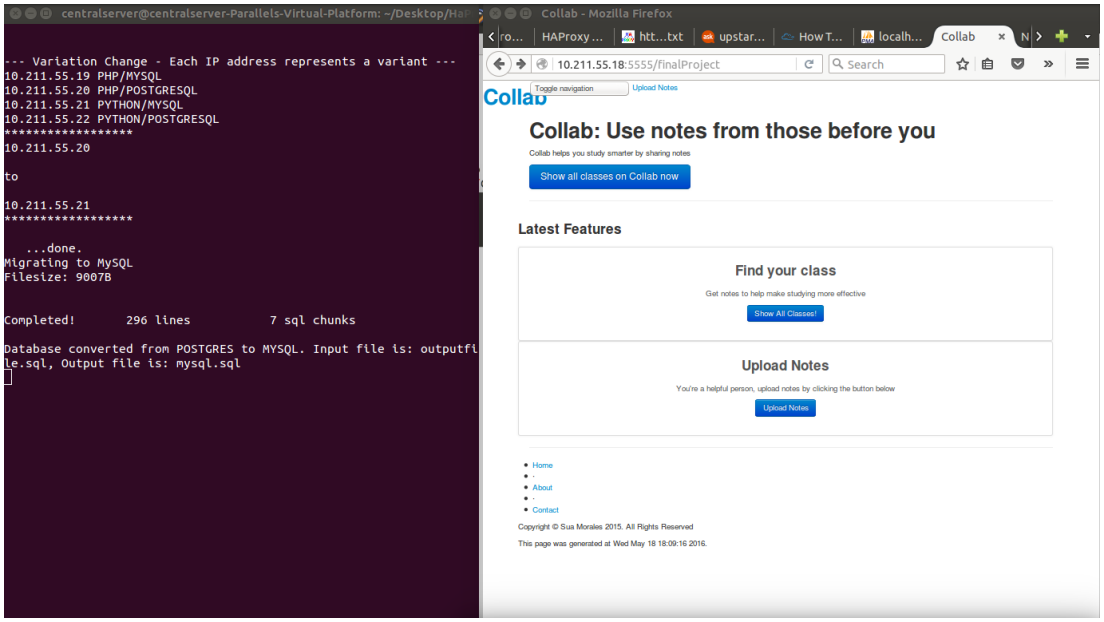

Figure A.1: PHP and PostgreSQL translated to Python and MySQL

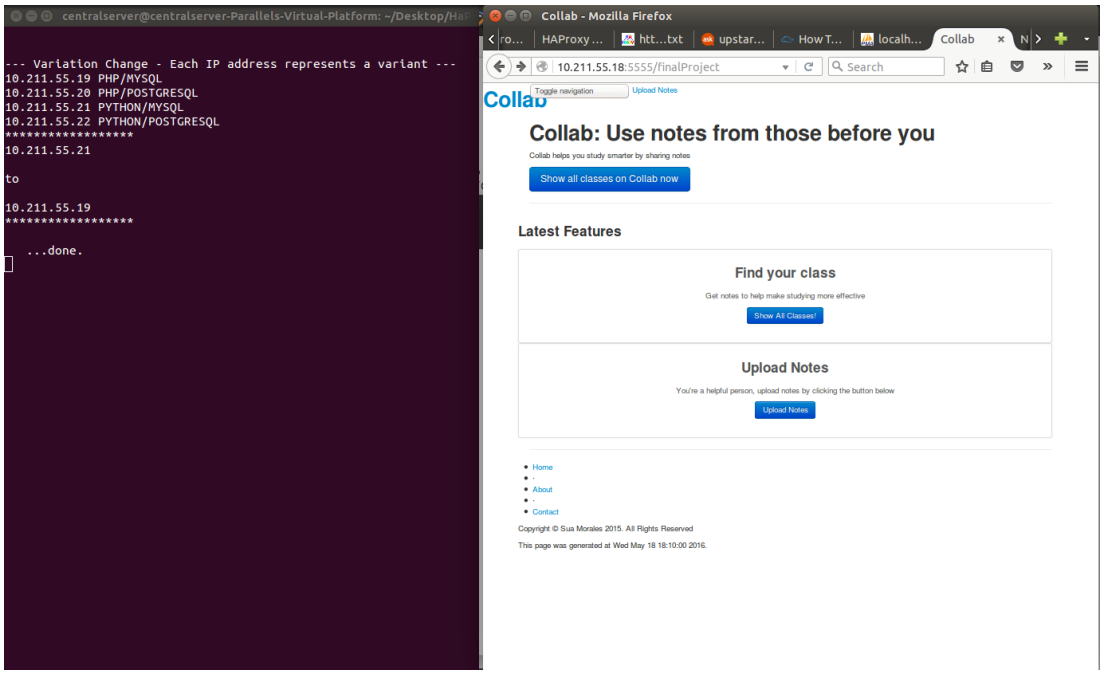

Figure A.2: PHP to Python translation with MySQL

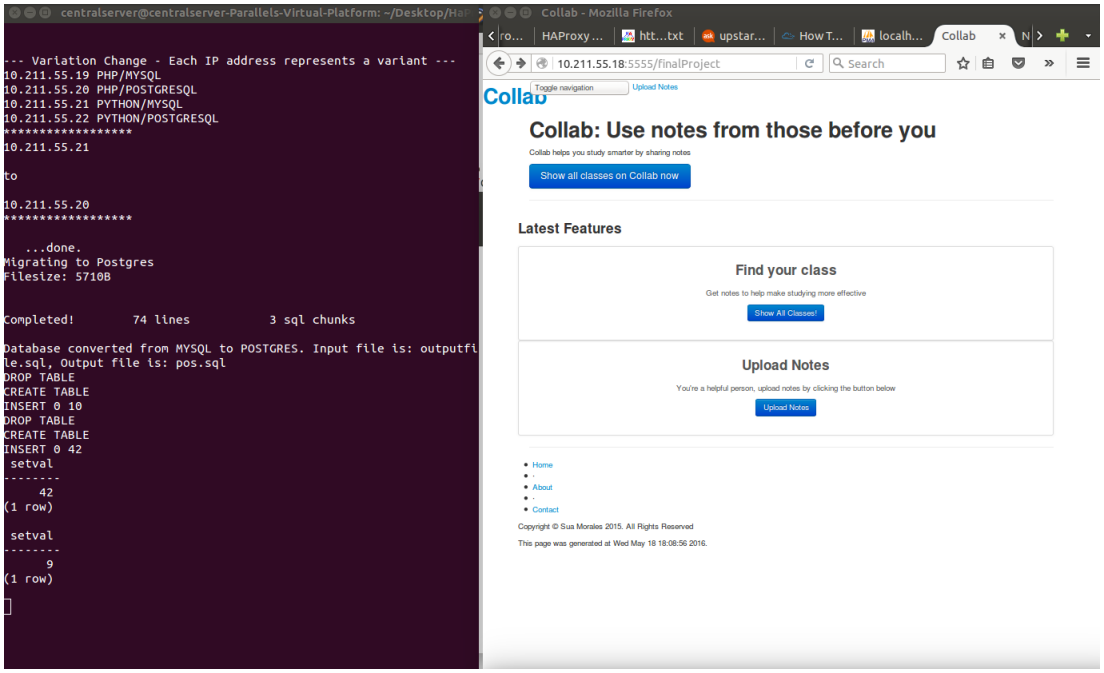

Figure A.3: Python and MySQL translated to PHP and PostgreSQL

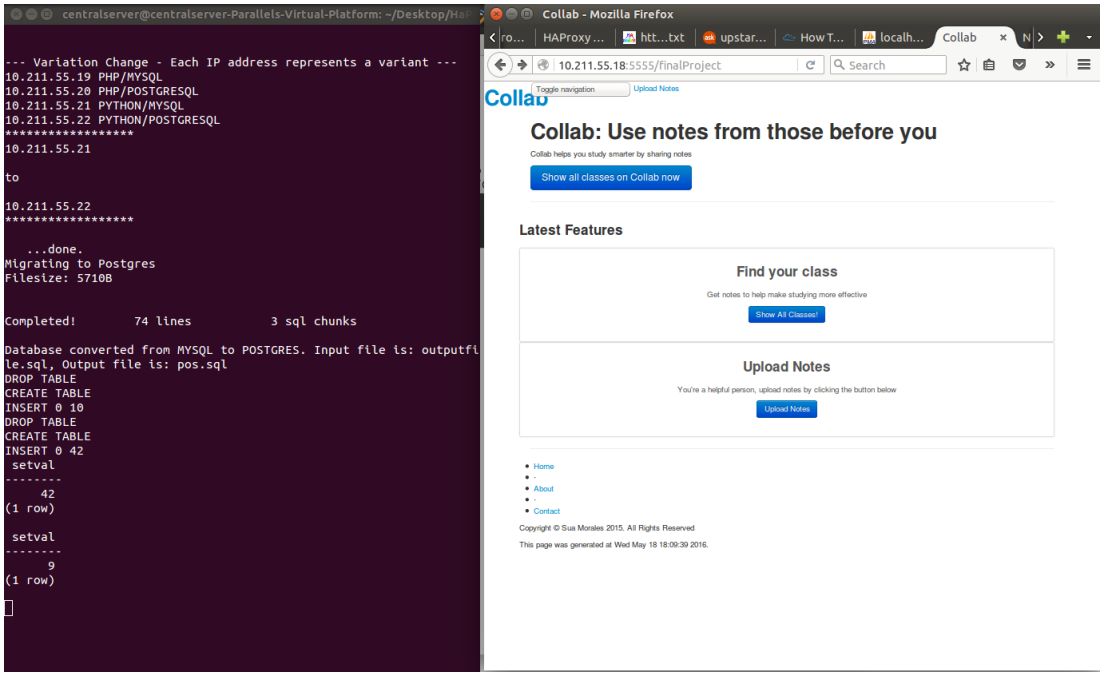

Figure A.4: MySQL to PostgreSQL translation with Python

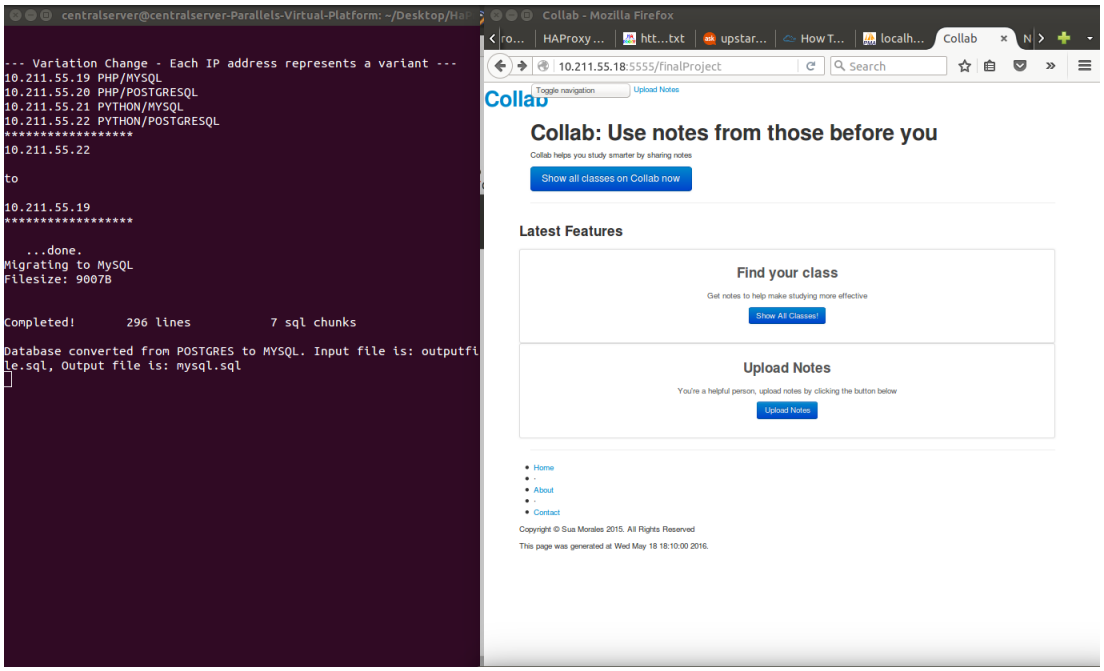

Figure A.5: Python and PostgreSQL translated to PHP and MySQL

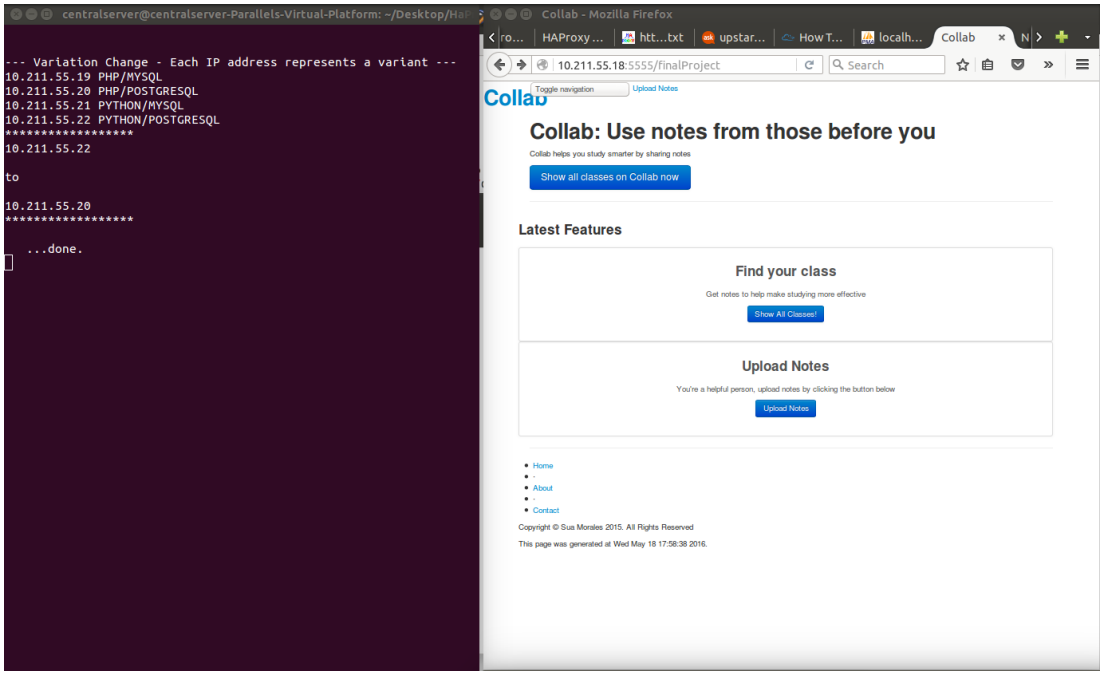

Figure A.6: Python to PHP translation with PostgreSQL

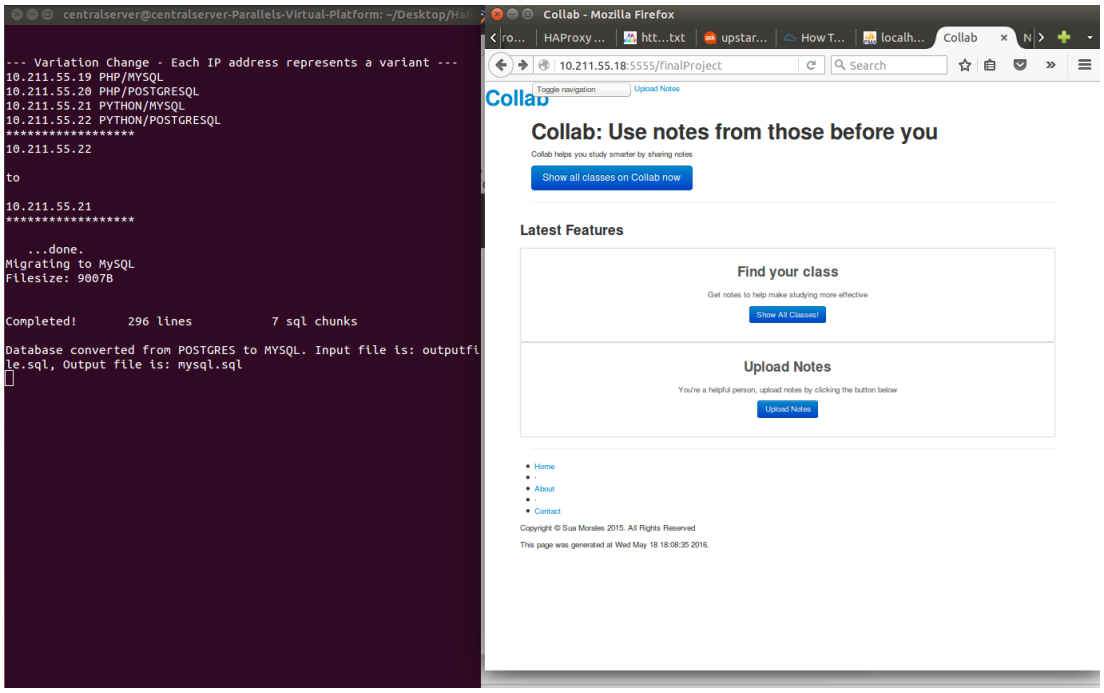

Figure A.7: PostgreSQL to MySQL Translation with Python

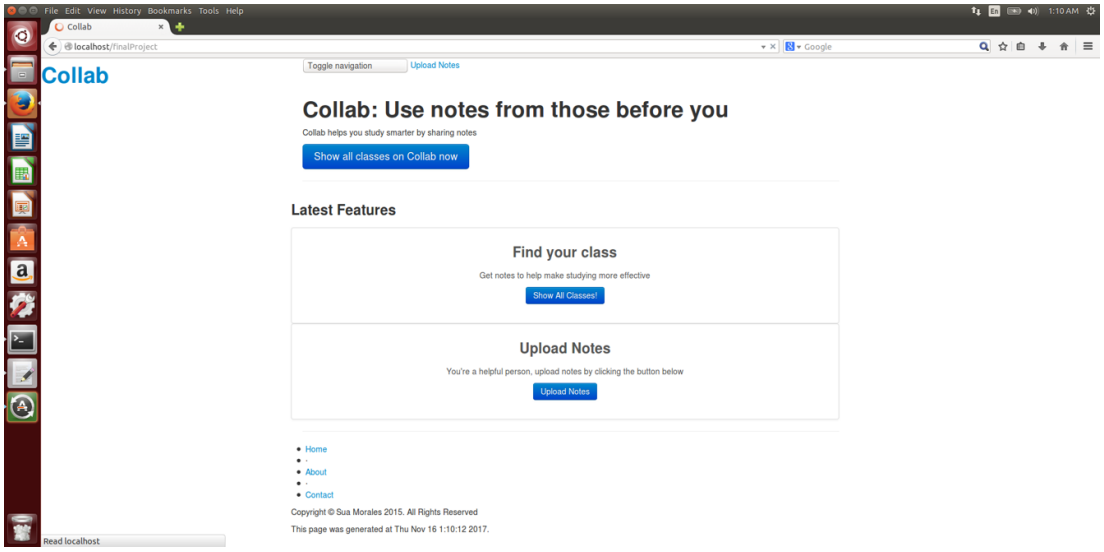

Figure A.8: Step one of uploading notes

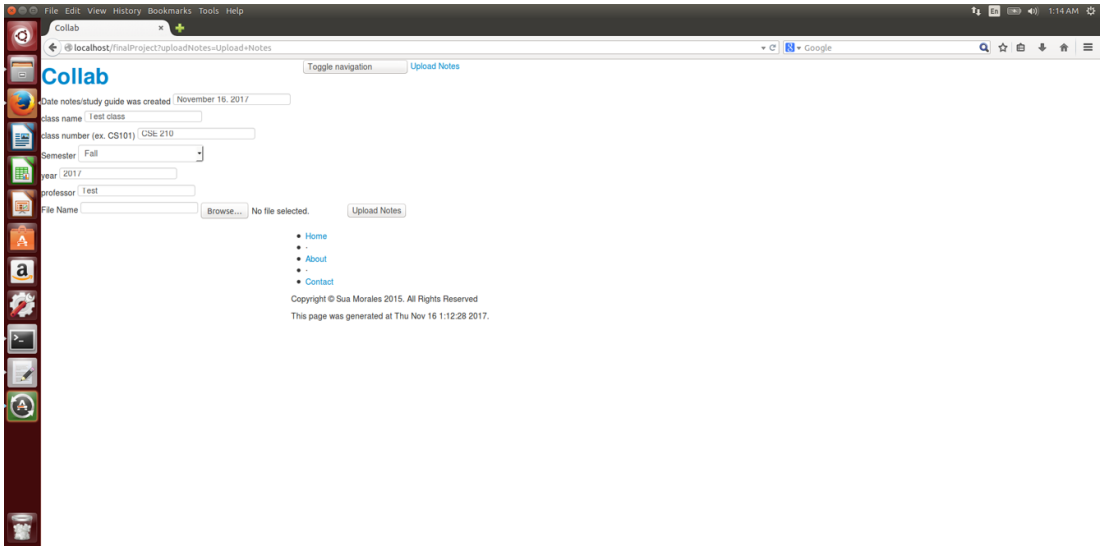

Figure A.9: Step two of uploading notes

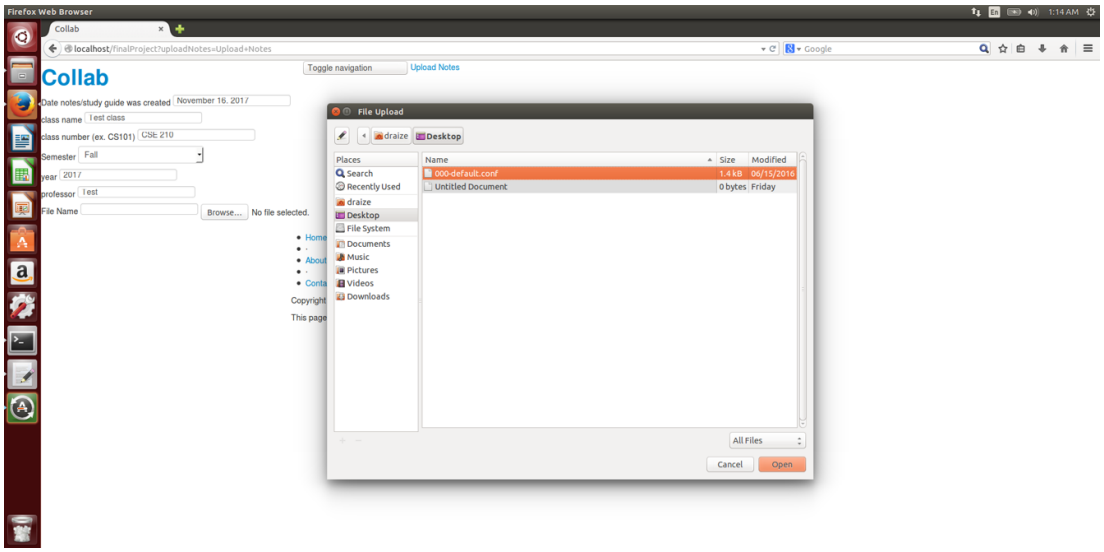

Figure A.10: Step three of uploading notes

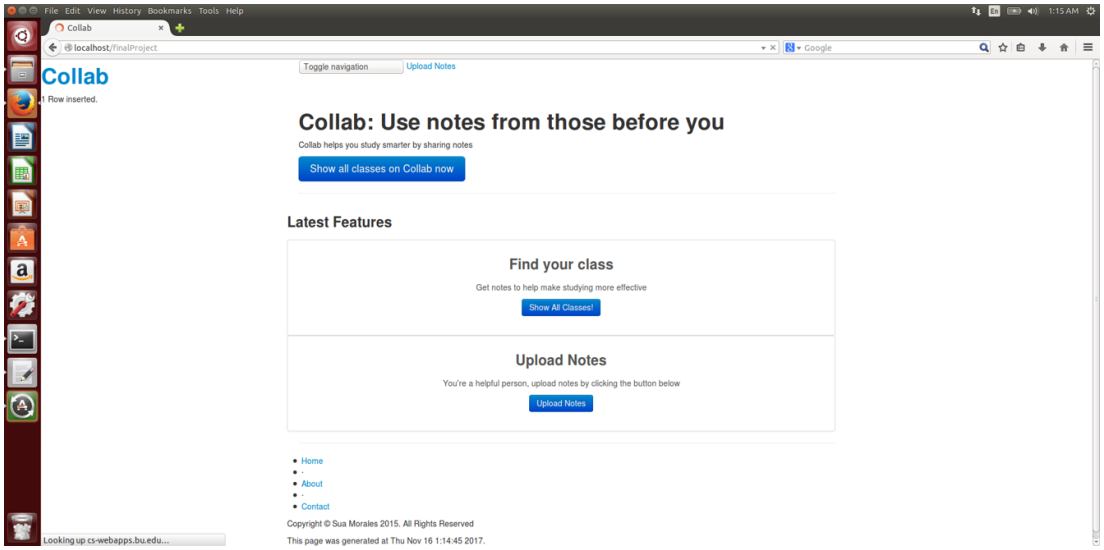

Figure A.11: Step four of uploading notes

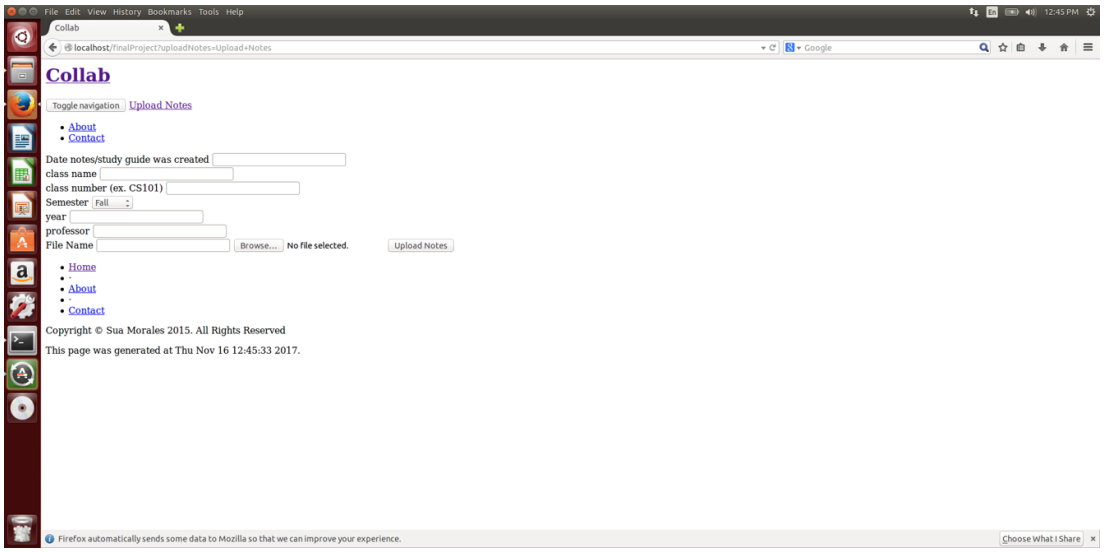

Figure A.12: Uploading note functionality post translation