University of Texas at Dallas

Dropbox Security

SYSM 6309 – SEM – Summer 2013 – Term Paper

Professor – Dr. Lawrence Chung

Lewis Scaife



**Abstract**

This paper discusses a security glitch that took place on June 19, 2011, which affected 25 million *Dropbox* users. From 1:54pm PT until 5:46pm PT, any *Dropbox* account was accessible to any other user with knowledge of the targeted user’s email address. *Dropbox* had implemented a code update which introduced a security flaw into the system. Any password string was being accepted and would authenticate an illegitimate user. In Requirements Engineering there are goals and problems, in the case of this paper, security is the goal. The problem is the fact that the code update introduced a vulnerability into the system. This paper will look back as to how this security issue happened and offer some processes and tools to help mitigate the same issue from happening again.

Confidentiality, Integrity, and Availability (C.I.A) are three pillars of security. This paper will discuss *authentication* and *access controls* which are subsets of C.I.A. In the case of *Dropbox* it is clear that the necessary controls and assessments were not implemented prior to release of the code update. *Dropbox* provides a service to its customer – storage of files/data. Many other cloud providers and web application developers offer services which must be managed through a Lifecycle. The IT Service Management Lifecycle is one of the important processes that this paper will touch on. Moreover, a methodology for mitigating authentication bypass and access control vulnerabilities in web applications will be explained. Nemesis is a shadow authentication system which can be used by companies such as *Dropbox* to prevent unauthorized access to user’s files in a cloud environment. It is the assertion that with a well-defined structured process and systems to verify identity/access, the 2011 security flaw at *Dropbox* would not have happened.

**1 Introduction**

*Dropbox* was founded in 2007 by two MIT students (Drew Houston and Arash Ferdowsi). In 2008 *Dropbox* received $7.2 million dollars from Sequoia Capital and other investors to further develop and market its service. *Dropbox* is a data storage company who utilizes the powerful computational power of cloud computing. Cloud computing builds off of the idea of virtualization, where systems have the following characteristics:

1. On Demand Self-Service
2. Broad Network Access
3. Resource Pooling
4. Rapid Elasticity
5. Measured Service

Cloud services must be reliable and available. The non-functional requirement of reliability, in regards to security was not satisfied. Advanced Requirements Engineering seeks to define what it means to have security. The requirements involved with security should be traceable to the design phase. Three questions were asked at the start of this case study:

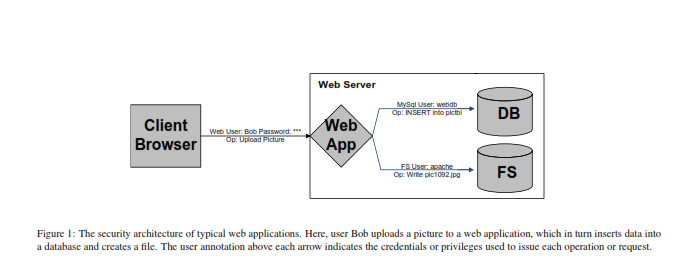
1. Were software requirements specified to the programmer or development team who applied the software patch? (Unknown)
2. Were quality controls in place during software development? (i.e. Regression Testing)
3. Did *Dropbox* test the patch against their development systems before deploying if to their production environment? (No)

Intensive research was done to determine what lines of code were responsible for the security glitch. In lieu of that code, it is assumed that an access check was missing or incorrectly performed in the web application code. This ‘bug’ or unexpected behavior resulted in *Dropbox* users and others being able to access any *Dropbox* directory without a valid password.

**2 Web Applications**

*Dropbox* is a web application. Users can logon via the internet and access their files almost anywhere. A Web applications simply allows users to access their data from any computer and interact and collaborate with one another. Each web application typically will deploy its own authentication and access control framework. This is exactly what *Dropbox* had done in an attempt to secure user data. It is often the case that once a user logs into a web application, an authoritative process/service acts on behalf of the authenticated user. The user does not have access rights to the storage location where the files are located, only the administrative process/service does. The Access Control checks, if user based, would have to explicitly check whether the requesting user is authorized to perform an operation. Rather than introduce these checks (based on security policies) throughout application code, the service account executes code with full privileges on behalf of user with limited privileges. This is where the security in web applications becomes flawed. Web application security is viewed as an instance of the *confused deputy problem* where the program can be fooled into misusing its authority.

When a user connects to a web application (i.e. *Dropbox*) they supply their username and password. The web application performs the security checks to ensure that username and password are valid. Once the credentials are validated, a login session is created for the user. The login session enables the user to not have to enter his/her password again when a new page is accessed. In order to keep track of this information either a cookie is used and installed on the user’s computer or authentication information is stored on the server and a cookie with a unique session ID is sent to the user.



**3 Attacks on Authentication and Access Control**

There are two types of vulnerabilities that can affect security:

**Authentication bypass –** an attacker can fool the application into treating his or her requests as coming from an authenticated user. An example of this is when an authentication state in an HTTP cookie is modified.

**Access Control Check** – this type of vulnerability occurs when an access check is missed or incorrectly performed in application code.

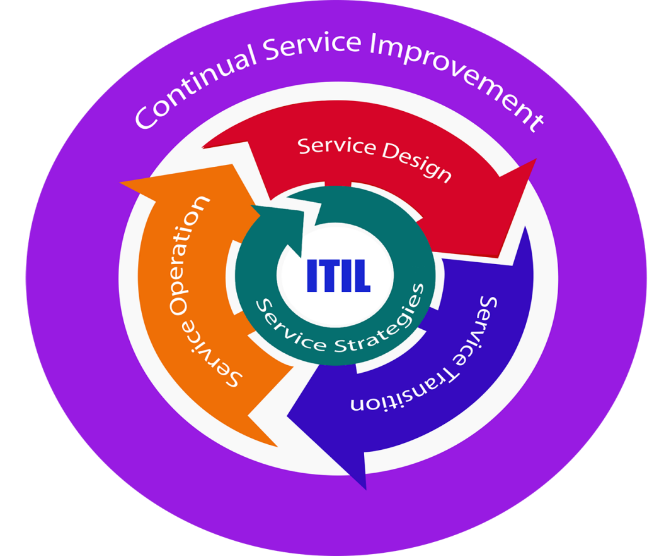
*Access control check* is the most likely reason why the *Dropbox* security glitch happened. A programmer/s could have developed code which did not check or verify settings in an access control list. When *Dropbox* users logged in the code accepted any password, because it did not call the function to verify the password field. Although authentication and access control bugs are more difficult to detect, because the code is application-specific. Also, developers must constantly refer to the security policy and previous framework used by previous developers.

**4 Information Technology Infrastructure Library (ITIL)**

ITIL is a framework that could have been used to catch this error before the application code went to production. This process along with a security audit would have assured that this error would have been found before it could cause disadvantageous results.

ITIL is a service management strategy composed of five stages:

* Service Strategy
* Service Design
* Service Transition
* Service Operation
* Continual Service Improvement



The *service strategy* defines plans for delivering and managing information services that meet an organization’s business needs. As seen in the above ITIL model, the service strategy is the core of the Service Lifecycle. The service design attempts to create IT systems that effectively achieve the goals of the service strategy. This phase provides guidance on the design of the IT services, processes, and service management. The *service transition* utilizes various processes, systems, and functions in order to build, test, and deploy a service. This stage attempts to ensure that the new service and subsequent changes are efficiently executed with minimal impact to clients. *Service operations* provide guidance on achieving the desired service levels after the service is made available to end-users. *Continual Service Improvement* (CSI) strives to improve service quality and effectives. This phase looks to identify services which are underperforming or performing as intended, and make improvements to those services on a continual basis. The CSI is made up of seven steps which *Dropbox* could have used to determine whether or not the code update/patch was going to perform as intended. The seven-step process is listed below:

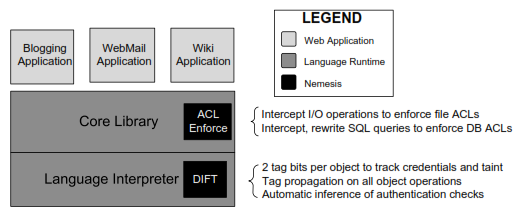
1. Define what you should measure.
2. Define what you can measure.
3. Gather the data.
4. Process the data, including checking for accuracy.
5. Analyze the data to determine if targets were met, understand trends, and relationships among data, and propose corrective action.
6. Present and use the information and corrective actions.
7. Implement corrective actions and improvements.

*Security Audits* are common checks that verify whether the security policy is satisfied before code/program is deployed into production. A security audit can be inserted into the ITIL framework to ensure reliability for security.

**5 Nemesis**

Nemesis is a security methodology which aims to address the problem of authentication in web applications without relying on the safety or correctness of the existing code. Nemesis functions as a shadow authentication system that works alongside of the existing application’s authentication framework. In order to determine how the web application authenticates users, Nemesis uses *Dynamic Information Flow Tracking* (DIFT) to track the flow of user credentials (i.e. user name and password), through application code. This functionality is combined with programmer-supplied access control rules on files and database entries. This ensures that only properly authenticated users are granted access to any privileged resources or data.

Nemesis prevents authentication and access control vulnerabilities by verifying when a user has been properly authenticated. It then uses authentication information to apply the necessary access controls on the users.



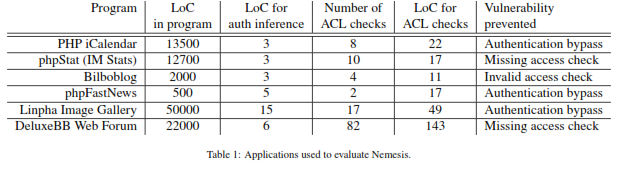
Nemesis also requires the developers/programmers to provide the location where the user names and passwords are stored. For third-party authentication frameworks, using LDAP for example, the external function invoked must be specified.

Nemesis keeps track of two bits for each data item in the application:

1. Credential bit
2. User Input bit

The credential bit contained whether or not a user supplied a known-good password or other credential. The user input bit contained all values submitted through an HTTP request (i.e. headers, cookies, POST bodies, and URL parameters. These bits can be stored per object or per byte.

Nemesis mitigates the attacks by connecting its shadow authentication system with an access control system protecting the web application’s database and file system. A prototype was developed and the results show that Nemesis was successful in preventing authentication bypass and access control check attacks. The results from testing were from five programs:



*Dropbox* could have benefited from ITIL and Nemesis, in mitigating the results of the software patch. The software patch introduced a ‘bug’ or unexpected behavior. It is the assumption that regression testing and security audits were not preformed. In closing the process and tool described within this paper should have prevented this problem.

Nemesis can also be used to improve precision in other security tools, such as those that find SQL injection bugs, by avoiding false positives for properly authenticated requests. Lastly, Nemesis only required less than 100 lines of code to be added to mitigate authentication and access controls vulnerabilities.

**References**

<http://blog.castsoftware.com/dropbox-drops-the-ball>

<http://news.cnet.com/8301-31921_3-20072755-281/dropbox-confirms-security-glitch-no-password-required/>

<http://www.huffingtonpost.com/2011/06/21/dropbox-security-bug-passwords_n_881085.html>

M.Dalton, C. Kozyrakis, and N. Zeldovich. Nemesis: Preventing Authentication and Access Control Vulnerabilities in Web Applications. In Proceedings of the USENIX Security Symposium, 2009.