SECURITY PITFALLS OF A SHARED PORTAL

LESSONS LEARNED FROM A CUSTOM PORTAL DEVELOPMENT PROJECT

Securus Global, March 2014

ABSTRACT

This paper sets out to examine the lessons learned from a client who commissioned a custom web portal to be developed. During our security assessment, we found a number of significant security vulnerabilities which lead to data theft, account take over and system compromise. The security pitfalls which lead to the compromises will be outlined along with the recommendations and strategies for avoiding these issues within your own projects.

Introduction

A client recently developed a custom web portal in-house to facilitate sensitive information sharing with external stakeholders. The purpose of the web portal was to provide a place where connected organisations could log in to view account details, agreements and other sensitive documents. A key intended design feature of the portal was to prevent one entity from viewing other entities' data. This data was considered sensitive as it contained special negotiated terms between the entities and was considered a trade secret.

Security Pitfalls

This security assessment highlighted a number of security pitfalls we tend to see being made by many organisations who develop custom software development today.

Some of the major pain points were caused by the web portal implementation using non-standard security methods. The project had tried to "roll their own" security mechanisms instead of using tried and tested existing methods. This is rarely a good idea! Implementation of security can be difficult to get right at the best of times - tried and tested security controls should always be used where they're available. Time and time again, we test custom security solutions that lead to security breaches.

Consistency is hugely important in security; it only takes one mistake for a web site to be compromised. The portal here was by and large well protected through input validation, however, three exceptions were found hidden throughout the portal, which allowed attackers to bypass controls.

Finally, another important take-away from this security assessment was the importance of maintaining good security at each layer. A slight misconfiguration within a single component can be taken advantage of in ways never intended by the system designers. In this case, a combination of both application and infrastructure vulnerabilities were combined in order to take full control of the web server.

Analysis

During the security assessment, it became clear that a large number of vulnerabilities existed within the client portal but a small number of network level misconfigurations were also found. When the two were combined, an attacker could extend their reach far beyond just the web portal.

Let's begin with the application layer, where a number of serious flaws within the client portal resulted in the unauthorised access of sensitive user data by other customers.

The client portal provided a reporting function, which allowed reports to be downloaded. However, this feature was implemented using a non-standard method where the name of the file being requested was simply obfuscated using base64-encoding. By modifying the request URL and changing the name of the file, it was possible to access any file on the web server, including configuration files and the client portal source code.

Uploading of files was also supported by the client portal. As a result, an anti-virus component was deployed. Surprisingly, the anti-virus protection only scanned individual files and was not configured to check archived zip files. As a result, it was possible to upload a virus to the system when included within a zip archive.

The file upload feature itself appeared at first glance to be implemented safely. However, after further investigation anomalous behaviour was found to exist in another form field, whereby if a file exceeded a certain size, it was uploaded and stored on the webserver's file system under the "web root" directory. This turned out to be a very critical issue which resulted in remote command execution on the web server.

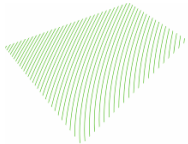
Against the intention of the client portal designers, a customer was able to view another customer's sensitive documents. Similar to the report download feature, a user was also able to download other sensitive documents that didn't belong to them. Again, base-64 encoding was used to mask the name of the requested file and it was possible to modify the request to obtain other customer's sensitive data.

It was also possible to take over the account of another customer. This was achieved through the account details page. When a customer clicks to view their account details, no authorisation checks were enforced. This was because the portal designers assumed that a customer would only ever be clicking on the links provided to them on-screen. But what if a malicious customer or an attacker decided to modify the link provided to them? Consequently, it was possible for a customer to modify the request and view other customers' account details. Taking this a step further, the email address could be changed and a password reset requested, resulting in a complete account takeover.

Validating user input is a basic yet extremely important security principle. In general, the input validation was found to be very good throughout the client portal. However, it only takes a single instance where such rules have been forgotten or ignored. In this case, two instances of SQL injection were found where parameters were concatenated together instead of using parameterised queries. Reflected Cross Site Scripting was also found within the application. Although the portal had very good input validation overall, it was let down by the lack of consistency in a couple of places. While as defenders, we need to find all unauthorised routes of entry, an attacker only needs to find one!

On the infrastructure side, insecure file permissions led to privilege escalation on the system and it was possible to become the "root" user on the web server. This was achieved by accessing the web server's directory and configuration file in combination with a weak sudo configuration. As a result, it was possible to create and run a program with "root" privileges, even though we only had access to a lower privileged account to begin with.

Finally, passwords were found to be stored insecurely and used a weak obfuscation algorithm. This resulted in two service accounts being compromised as well as the Unix web administration password which could be seen in cleartext inside the command history file.



Lessons Learned

Overall, we discovered a number of serious security vulnerabilities during our assessment, but what can we learn from this and how did the client remediate these issues before launching the portal live on the internet?

- One of the most significant lessons learned was to always use standard security methods and practices. For example, always use your language or framework's security features. Instead of trying to hide filenames and usernames with base-64 encoding, use the programming language's standard API for encryption. Of course, it is always preferable to avoid sending such data at all if possible, however, in this case it was not possible to avoid.

- Never assume that a user is who they say they are without checking each time. The lack of authorisation checks caused a number of issues for the client portal. Specifically, the ability to take over another customer's account and view other customers' documents. A session identifier is assigned to each logged in user, this should be checked for authorisation on the server side before any sensitive data is transmitted.

- Never trust any data submitted by a user. In general, input validation was found to be good throughout the client portal. However, as we saw, it only takes one forgotten field or parameter on the site to open up the site to exploitable attacks. Another input validation related issue arose during the file upload, where an in-secure behaviour was found when a particular value was submitted with unexpected data. This turned out to be a very critical issue, which led to remote command execution against (and complete compromise of) the web server.

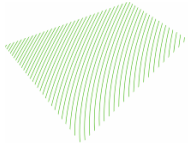
- Always thoroughly test and review configuration files. The anti-virus component was configured to ignore zip files and this should have been found through a configuration review and execution of standard test cases.

- Do not rely on other layers to provide security; always secure each layer to be as strong as possible as part of a defence in depth strategy. The security vulnerabilities within the application layer were further compounded by the security misconfigurations at the system infrastructure layer. Since the file permissions and history file were not configured securely it was possible to execute commands as the "root" user and obtain the administration password.

Business benefits

Penetration testing is a vital part of a healthy software development lifecycle. The earlier bugs are caught the easier and more cost efficient they are to fix. This is why we often have clients request a penetration test early in the development cycle and again before go-live. We are also often engaged to provide training in security best practices to both architects and developers to assist with integrating security through the SDLC.

While a penetration test before go-live is essential, it should not be relied upon as the only security gate within the project roadmap. Instead, a security mindset should be integrated into all parts of the software life-cycle, from design to deployment. The importance of security needs to come from senior executives downwards in order to build a supportive security culture. Security should be built into the project plan to allow designers and developers time to design and implement security thoroughly. While we specialise in high quality penetration testing, we really feel we have the most impact through our secure developer training where many security vulnerabilities can be avoided completely before any penetration testing is scheduled.



Summary

In summary, many software security vulnerabilities can be avoided earlier in the software development lifecyle through increased developer training and support. Security is a very different mindset to feature development and performance enhancements which developers are expected to produce under heavy time pressures each day. The cost of a security vulnerability has been estimated to increase 30 times if only discovered at the end of the lifecycle compared with being identified at the design phase.

We strongly recommend periodic developer training, this combined with the lessons learned from penetration testing which can be fed back into the software development lifecycle to produce continued improvements to the security of your code. After all, security is a mindset and not just an individual product or assessment.

More Information

Please reach out if you would like more information on web portal issues and solutions as they relate to your organisation.

T: +61 (0) 2 9283 0255
E: info@securusglobal.com
www.securusglobal.com