Pentest

Date 03/03/2016

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Customer Information | | | | | |
| Company Name: |  | | | | |
| City: |  | State: |  | Zip Code: |  |
| URL: |  |  |  |  |  |

|  |  |
| --- | --- |
| Customer Contact Information | |
| Contact Name: |  |
| Title: |  |
| Telephone: |  |
| E-mail: |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Consultant Information | | | | | |
| Company Name: | Serpico Template Report, LLC | | | | |
| Contact Name: |  | | | | |
| Title: |  | | | | |
| Telephone: |  | | | | |
| E-mail: |  | | | | |
| Business Address: | 123 Paper St | | | | |
| City | TestCity | State: | MA | Zip Code: | 11111 |
| URL: | <http://www.serpicoreport.com> | | | | |

# Executive Summary

Serpico Template Company (STC) was contracted to perform a penetration test for . This report discusses the results from the assessment. Really, if you are reading this you should update the template to match your executive summary. The symbols throughout this report are used to display the data. Please see the README to understand how they work.

Overall, STC was able to achieve the goals of the assessment and exfiltrate the targeted data. There were a number of critical findings during the assessment including the following:

|  |  |
| --- | --- |
| Finding Name | DREAD Score |
| SQL Injection | 50 |
| Cross Site Scripting (XSS) | 46 |
| Direct Object References | 44 |
| XML External Entity (XXE) Processing | 50 |

MOAR STUFF

# Attack Narrative

The following discusses the details of the assessment.

# Findings

## Findings Table

The following were the results from the assessment:

|  |  |  |
| --- | --- | --- |
| Finding Name | DREAD Score | Remediation Effort |
| **High Risk Findings (36-50)** |  |  |
| SQL Injection | 50 | MEDIUM |
| Cross Site Scripting (XSS) | 46 | LOW |
| Direct Object References | 44 | LOW |
| XML External Entity (XXE) Processing | 50 | LOW |
|  |  |  |
| **Moderate Risk Findings (20-35)** |  |  |
| Path Traversal | 35 | MEDIUM |
|  |  |  |
| **Low Risk Findings (1-19)** |  |  |
|  |  |  |

|  |  |
| --- | --- |
| DREAD Score |  |
| Damage Potential | 10 |
| Reproducibility | 10 |
| Exploitability | 10 |
| Affected Users | 10 |
| Discoverability | 10 |
| Total | 50 |

### Summary

The OWASP guide [1] gives the following description for SQL Injection:

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.

### Proof

### Remediation

The following is recommended to prevent SQL Injection:

* Use of Prepared Statements (Parameterized Queries)
* Use of Stored Procedures
* Never trust user input, Escaping all User Supplied Input

|  |  |
| --- | --- |
| DREAD Score |  |
| Damage Potential | 8 |
| Reproducibility | 10 |
| Exploitability | 8 |
| Affected Users | 10 |
| Discoverability | 10 |
| Total | 46 |

### Summary

The OWASP guide [1] gives the following description for Cross-Site Scripting:

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted web sites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

An attacker can use XSS to send a malicious script to an unsuspecting user. The end user’s browser has no way to know that the script should not be trusted, and will execute the script. Because it thinks the script came from a trusted source, the malicious script can access any cookies, session tokens, or other sensitive information retained by the browser and used with that site. These scripts can even rewrite the content of the HTML page.

### Proof

### Remediation

The following is recommended to remediate XSS vulnerabilities:

* Never trust user input
* Never insert untrusted data except in allowed locations
* HTML escape before inserting untrusted data into HTML element content
* Use whitelists in place for Black lists for input filtering

|  |  |
| --- | --- |
| DREAD Score |  |
| Damage Potential | 8 |
| Reproducibility | 10 |
| Exploitability | 8 |
| Affected Users | 10 |
| Discoverability | 8 |
| Total | 44 |

### Summary

The OWASP guide [1] gives the following description for Insecure Direct Object Reference:

Applications frequently use the actual name or key of an object when generating web pages. Applications do not always verify the user is authorized for the target object. This results in an insecure direct object reference flaw. Testers can easily manipulate parameter values to detect such flaws and code analysis quickly shows whether authorization is properly verified.

### Proof

### Remediation

Use per user or session indirect object references. This prevents attackers from directly targeting unauthorized resources. For example, instead of using the resource’s database key, a drop down list of six resources authorized for the current user could use the numbers 1 to 6 to indicate which value the user selected. The application has to map the per-user indirect reference back to the actual database key on the server.

Check access. Each use of a direct object reference from an untrusted source must include an access control check to ensure the user is authorized for the requested object.

|  |  |
| --- | --- |
| DREAD Score |  |
| Damage Potential | 7 |
| Reproducibility | 7 |
| Exploitability | 7 |
| Affected Users | 7 |
| Discoverability | 7 |
| Total | 35 |

### Summary

Quoting from [1], a Path Traversal attack aims to access files and directories that are stored outside the web root folder. By browsing the application, the attacker looks for absolute links to files stored on the web server. By manipulating variables that reference files with “dot-dot-slash (../)” sequences and its variations, it may be possible to access arbitrary files and directories stored on file system, including application source code, configuration and critical system files, limited by system operational access control. The attacker uses “../” sequences to move up to root directory, thus permitting navigation through the file system.

This attack can be executed with an external malicious code injected on the path, like the Resource Injection attack. To perform this attack it’s not necessary to use a specific tool; attackers typically use a spider/crawler to detect all URLs available.

This attack is also known as “dot-dot-slash”, “directory traversal”, “directory climbing” and “backtracking”.

### Proof

### Remediation

|  |  |
| --- | --- |
| DREAD Score |  |
| Damage Potential | 10 |
| Reproducibility | 10 |
| Exploitability | 10 |
| Affected Users | 10 |
| Discoverability | 10 |
| Total | 50 |

### Summary

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, port scanning from the perspective of the machine where the parser is located, and other system impacts.

### Proof

### Remediation