

WebGoat vulnerability assessment report

Submission by: Goutham Vyasroimath Keshava Murthy

Assignment: AEP CS - Application and Web Application Security

Cohort: PGP AEP CS May 2022 Cohort



June 28, 2022

WHITE HATS organization

White Hat Street, White Hat State, White Hat Nation - 007007

Contents

[1. Abstract 2](#_Toc107317571)

[2. In-Scope 2](#_Toc107317572)

[3. Out-Of-Scope 2](#_Toc107317573)

[4. Executive Summary 2](#_Toc107317574)

[5. Definitions - Impact & exploitation probability 2](#_Toc107317575)

[6. Probability of exploitation 3](#_Toc107317576)

[7. WebGoat Vulnerabilities 3](#_Toc107317577)

[7.1. Vulnerability 4](#_Toc107317578)

[7.1.1. Finding 4](#_Toc107317579)

[7.2. Vulnerability 5](#_Toc107317580)

[7.2.1. Finding 5](#_Toc107317581)

[7.3. Vulnerability 7](#_Toc107317582)

[7.3.1. Finding 7](#_Toc107317583)

[7.4. Vulnerability 7](#_Toc107317584)

[7.4.1. Finding 8](#_Toc107317585)

[7.5. Vulnerability 8](#_Toc107317586)

[7.5.1. Finding 9](#_Toc107317587)

# Abstract

The objective of this report is to find web application vulnerabilities of WebGoat hosted on a VM Linux machine by using manual SQL Injection techniques. A thorough investigation of SQL Injection vulnerabilities was conducted. The impacts and probability of the exploitation of each vulnerability found was graded based on the standard vulnerability scoring scheme. Additionally, the vulnerabilities were graded according to the consequences of the exploitation identified through the findings. This report helps web application developers, providers and all relevant stakeholders on business & product side to better understand the inherent and possible security issues of WebGoat. This report assumes follow up corrective actions by the WebGoat team in implementing protection against reported SQL Injection security vulnerabilities

# In-Scope

Investigate and report SQL Injection vulnerabilities

# Out-Of-Scope

Recommendation to fix the SQL Injection vulnerabilities

# Executive Summary

WebGoat application has implemented client-side validations only. This has made pages accessible to parameter tampering through user input of which the attacker can take advantage. This attacker was able to get more information from system and ended up making updates through SQL Injection which exposes SQL Injection security vulnerabilities of WebGoat application.

# Definitions - Impact & exploitation probability

|  |  |
| --- | --- |
| **Consequence of exploitation** | **Exploitation** |
| Low | Even if the attack is successful, the attacker would not have control  over what’s being exploited, modification of the contents is not  also, possible. Hence, the attack impact scope may be limited to  only allowing an attacker to access a specific level of only viewing  the system contents. |
| Medium | The attack needs some pre-conditions for the attacker to exploit the  whole system because the system is configured with some security  measures. Therefore, a successful attack may allow little  modifications on the specific system contents which may not affect  the entire system. |
| High | There is a total compromise of the whole system, and therefore,  loss in system protections, which leads to full system files and  configuration disclosure. Moreover, an attacker can modify or  control the entire system. |

# Probability of exploitation

|  |  |
| --- | --- |
| **Consequence of exploitation** | **Exploitation** |
| Low | There is a need for sophisticated tools to bypass systems  restrictions to expose the vulnerability before starting to exploit the  system. So, this vulnerability is very unlikely to occur. |
| Medium | The chance for this kind of vulnerability to occur requires some  preconditions. So, this vulnerability occurs moderately. |
| High | For this vulnerability to be exposed, require only some open  available tools with some little knowledge. So, this weakness is  very common to occur |

# WebGoat Vulnerabilities

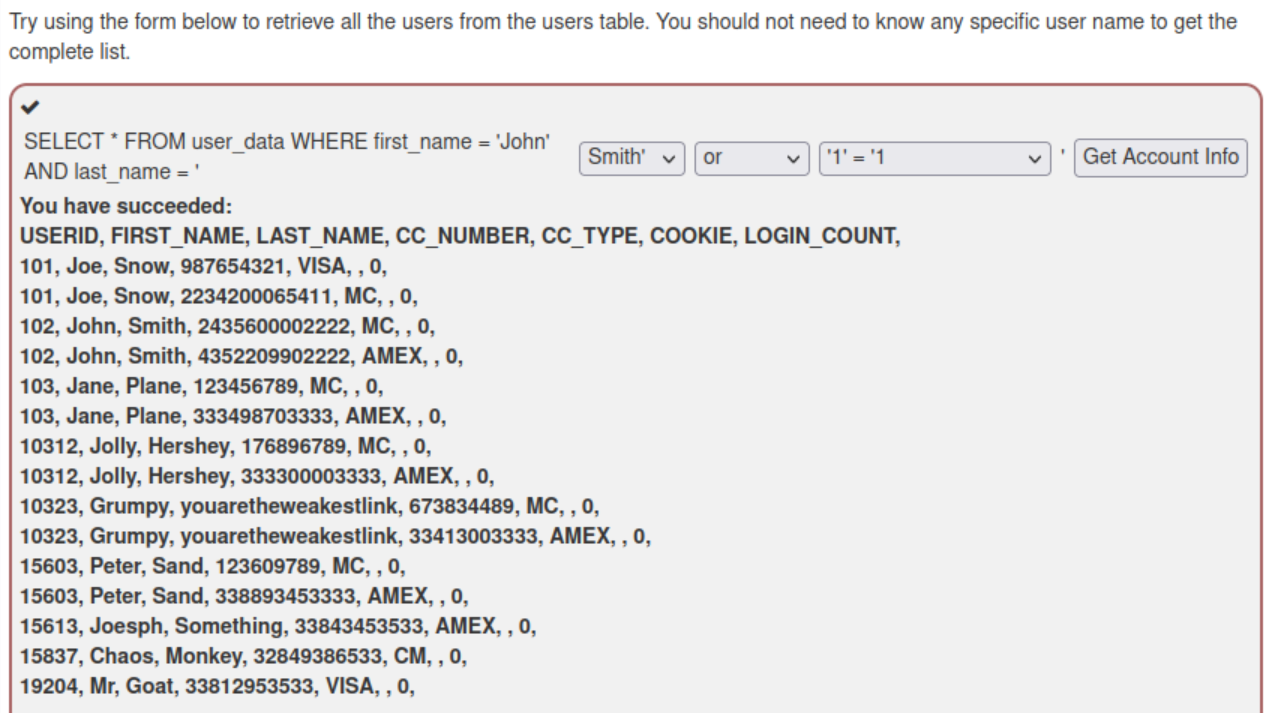
# Vulnerability

|  |  |
| --- | --- |
| **SQL Injection** | SQL String Injection |
| **Web Goat question reference number** | Question 9 |
| **Impact** | High |
| **Probability** | High |

# Finding

The user data is exposed as shown below to SQL String Injection attack. On the page by using strings like Smith’ or ‘1’=’1 the web application provided data of all the rows in user\_data thus exposing user details of all users

Smith’ or ‘1’=’1



# Vulnerability

|  |  |
| --- | --- |
| **SQL Injection** | SQL Numeric Injection  SQL String Injection |
| **Web Goat question reference number** | Question 10 |
| **Impact** | High |
| **Probability** | High |

# Finding

The query in the below examples is built by concatenating number which makes it susceptible to SQL Numeric Injection. At the same time, it is also susceptible for SQL String Injection attack. On the page by using inputs shown in scenarios below, the web application provided data of all the rows in user\_data thus exposing user details of all users

Scenario 1)

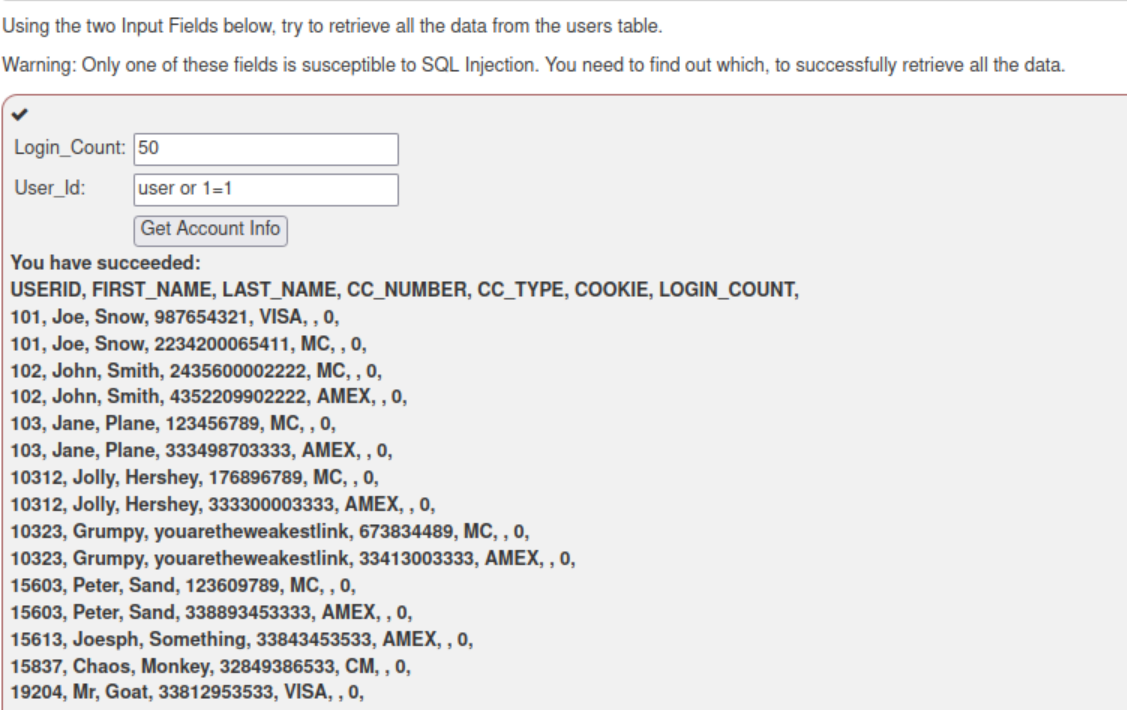
100

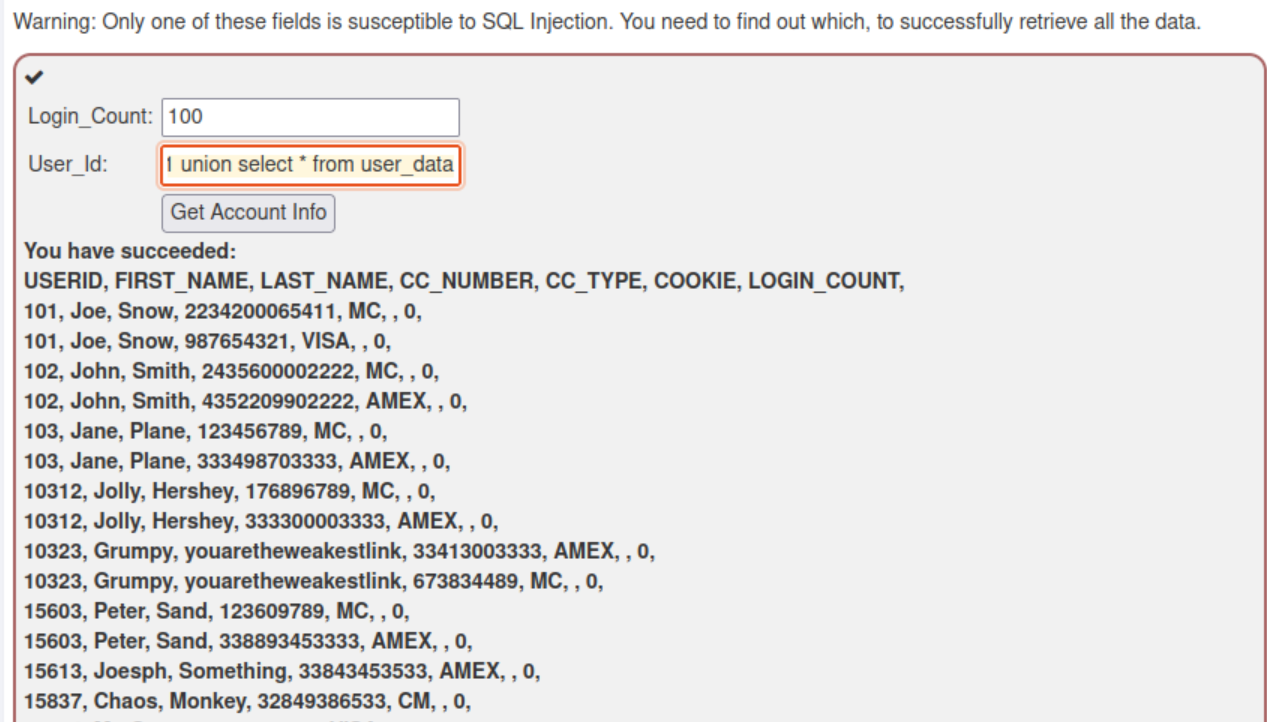
user or 1=1

Scenario 2)

100

1 union select \* from user\_data





# Vulnerability

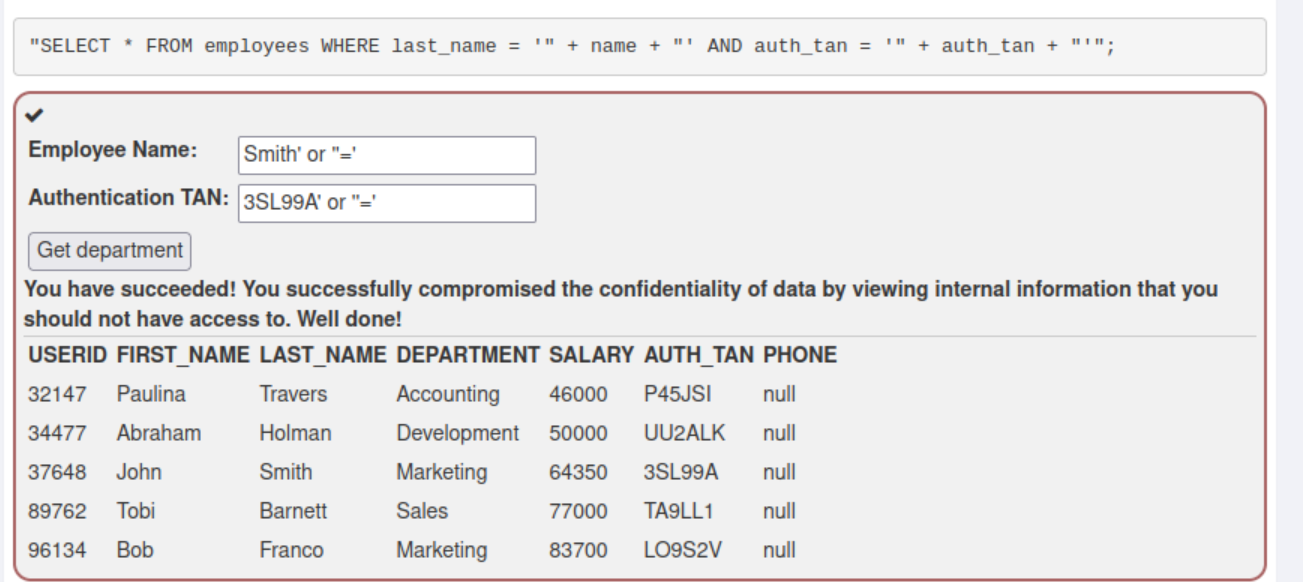
|  |  |
| --- | --- |
| **SQL Injection** | SQL String Injection – compromising confidentiality by query chaining |
| **Web Goat question reference number** | Question 11 |
| **Impact** | High |
| **Probability** | High |

# Finding

Employees salary data is exposed as shown below to SQL String Injection attack. On the page by using strings like Smith’ or ‘’=’ and 3SL99A’ or ‘’=’ the web application provided data of all the rows in employees table thus exposing salary information of all users

Smith' or ''='

3SL99A' or ''='



# Vulnerability

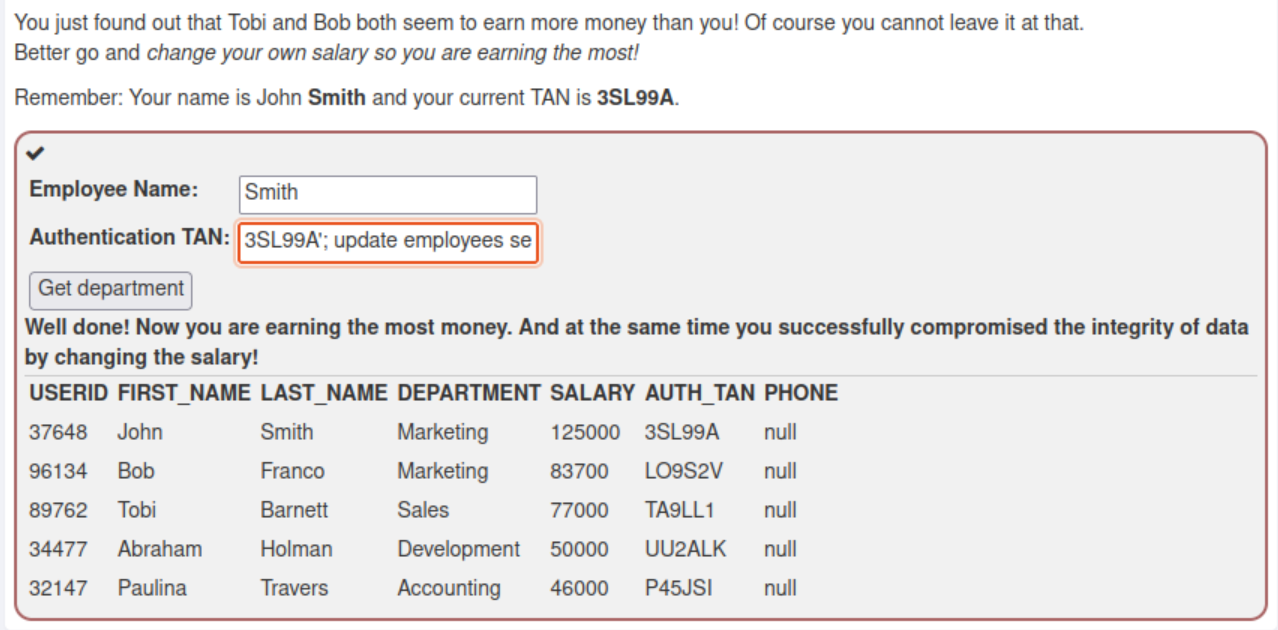
|  |  |
| --- | --- |
| **SQL Injection** | SQL String Injection – compromising integrity |
| **Web Goat question reference number** | Question 12 |
| **Impact** | High |
| **Probability** | High |

# Finding

Specific employee salary is updated in spite of employee not having the privilege to do it. After the employee salary is exposed, through manipulation of query chaining, a specific user is able to update salary for the user. Or even for others. By using query chaining as shown below, employee updates salary thereby compromising integrity of data

Smith

3SL99A'; update employees set SALARY=125000 where AUTH\_TAN='3SL99A'----



# Vulnerability

|  |  |
| --- | --- |
| **SQL Injection** | SQL String Injection – compromising availability |
| **Web Goat question reference number** | Question 13 |
| **Impact** | High |
| **Probability** | High |

# Finding

After compromising confidentiality & integrity, below shown vulnerability even allows user to delete data by dropping table. This is impacting availability aspect of application. After manipulating data, user is also able to drop table by query chaining as shown below

na'; drop table access\_log—

