

DEFENSIVE PROGRAMMING

Lecture for EDA 263

Magnus Almgren

Department of Computer Science and
Engineering

Chalmers University of Technology

Traditional Programming

When writing a program, programmers typically focus on what is needed to solve whatever problem the program addresses.

(p.391, Stallings/Brown)

- Common assumptions:
 - inputs a program will receive,
 - environment the program runs in,
 - a “cooperative” user, etc.

Defensive Programming

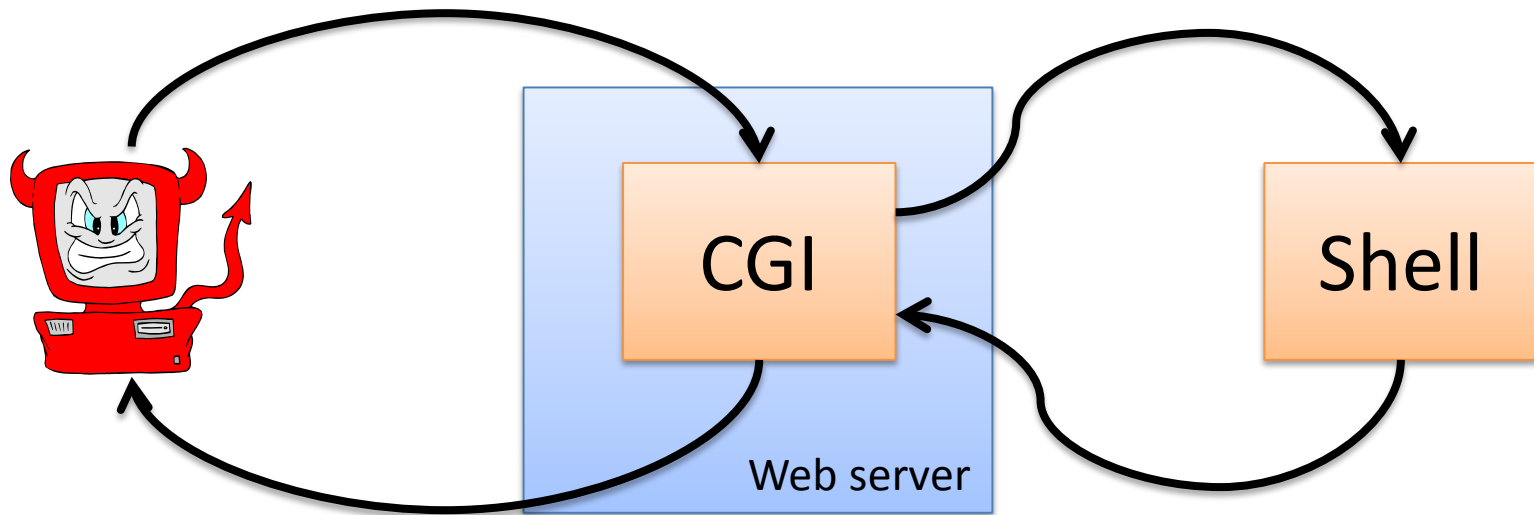
One should always take care when writing a program – code is reused and it is impossible to foresee how and when a module will be used in the future. ***Never trust user input!***

- Defensive programming / Secure Programming:
 - must always validate assumptions (nothing is assumed),
 - needs an awareness of the consequences of failures, and
 - the techniques used by attackers.
- Range of similar vulnerabilities exploited over time (CERT)
 - Injection Attacks (ex 12.2)
- Examples
 - Databases: part of almost all real systems
 - Web apps: often done quickly by junior programmers, but accessible by anyone in the world (see OWASP list)

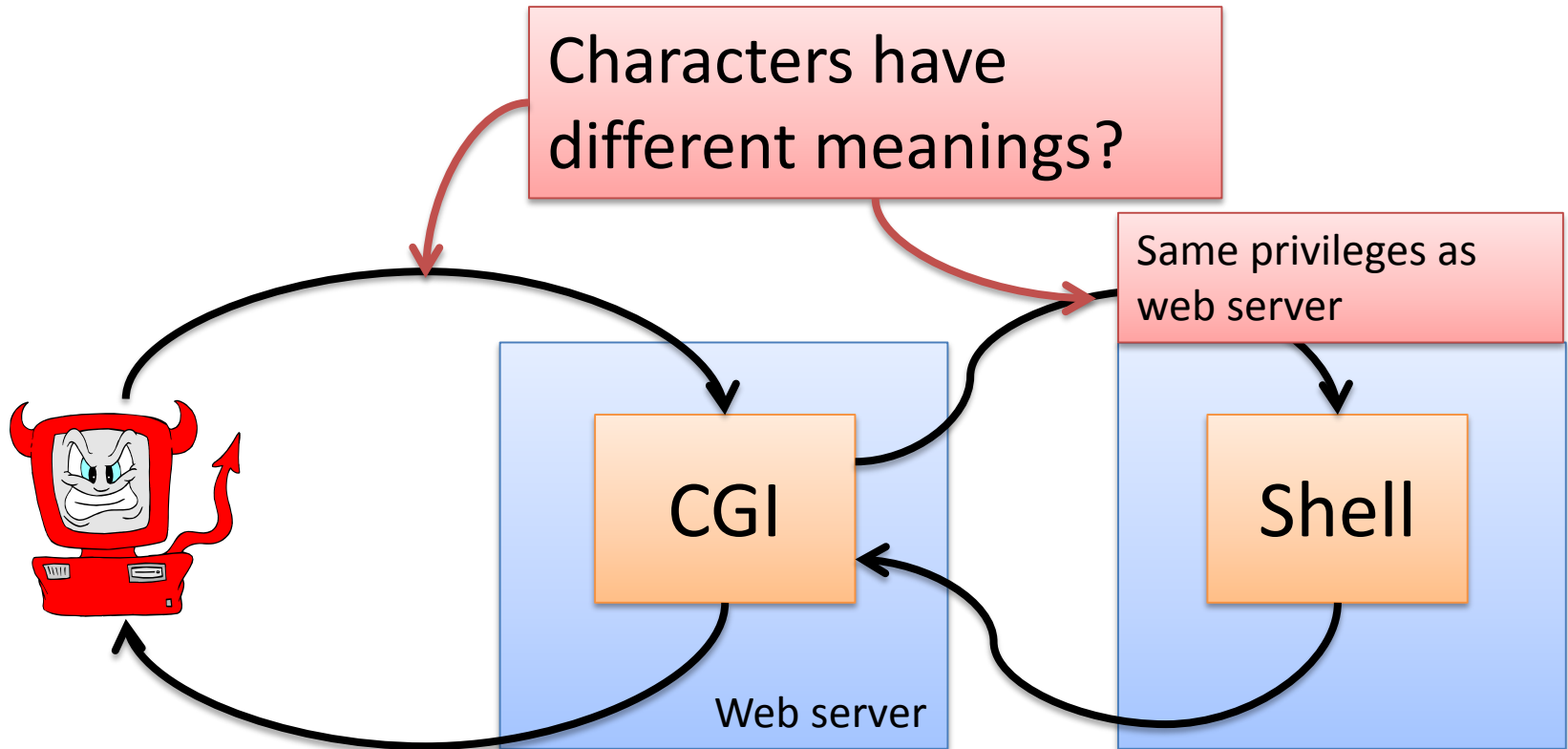
Domains

- Handling Program Input
 - Buffer Overflows
 - Injections Attacks
- Writing Safe Program Code
- Interacting with OS and other programs
- Handling Program Output

Example: Command Injection

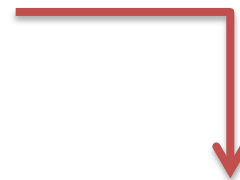


Example: Command Injection



Example: Command Injection Mitigation Technique

- Define what is known dangerous input
- Define what is valid input
- Problems:
 - Definition of what is really dangerous
 - Multiple encodings
 - For web: space = %20, / = %2F, ; = %3B
- Not only for strings but also other types of data: *integer overflows*

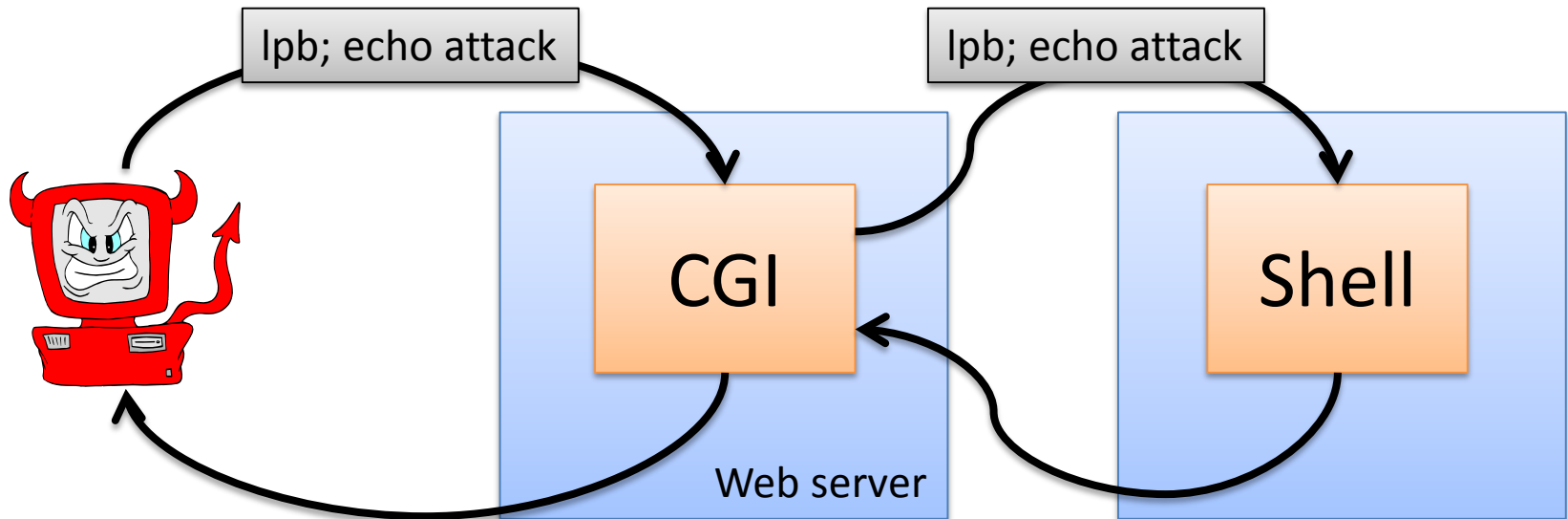


Vulnerability Note VU#20276

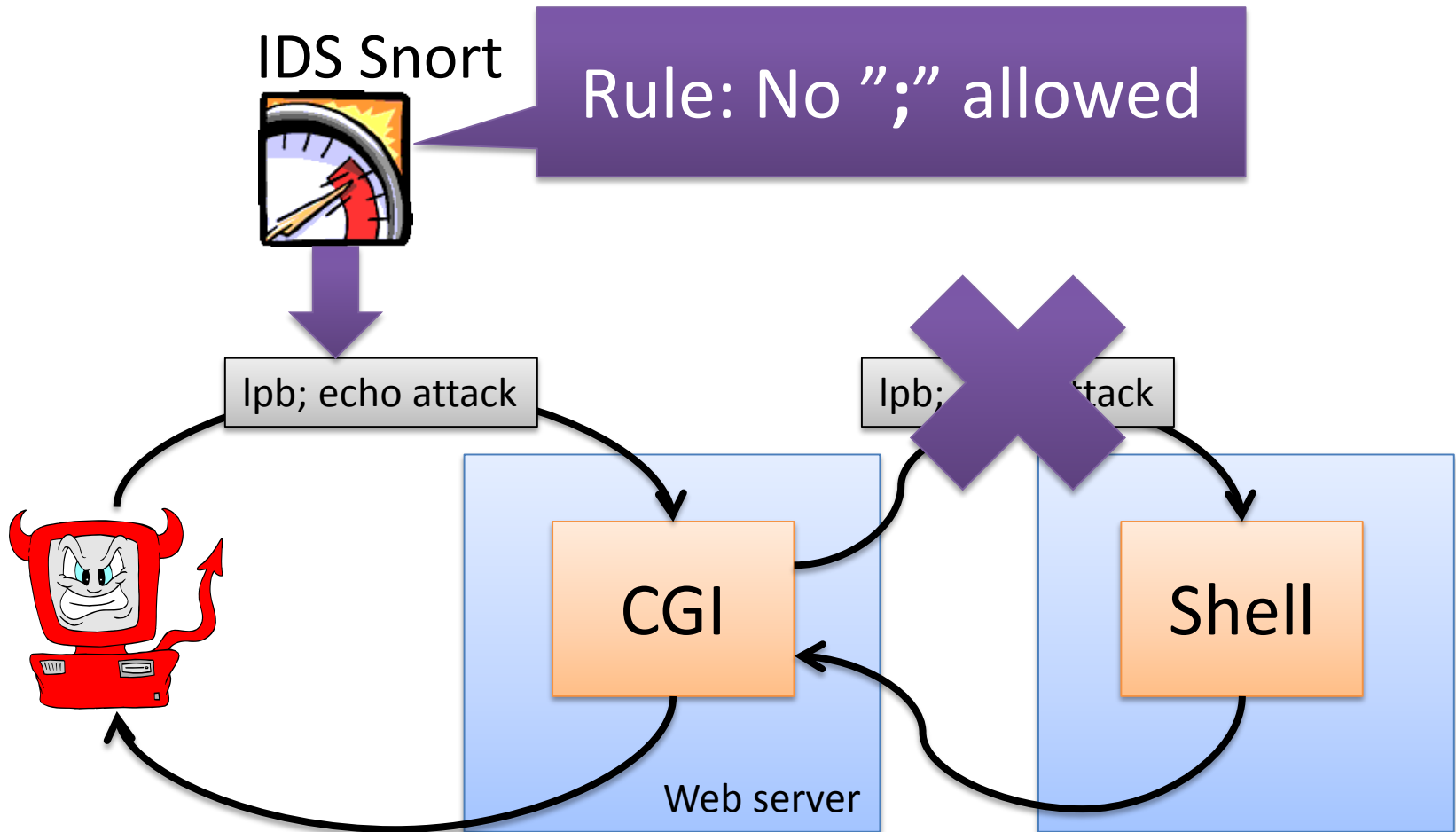
phf CGI Script fails to guard against newline characters

- CGI script phf, exploited late 1990's.
- Tried to sanitize input using a routine supplied in the web server
 - `escape_shell_cmd()`
 - Removing a number of shell meta characters BUT ...
 - Buggy: forgot to remove ***newline character***
- Attack:
embed a newline character in the string passed to the CGI script phf, resulting in additional commands to execute.

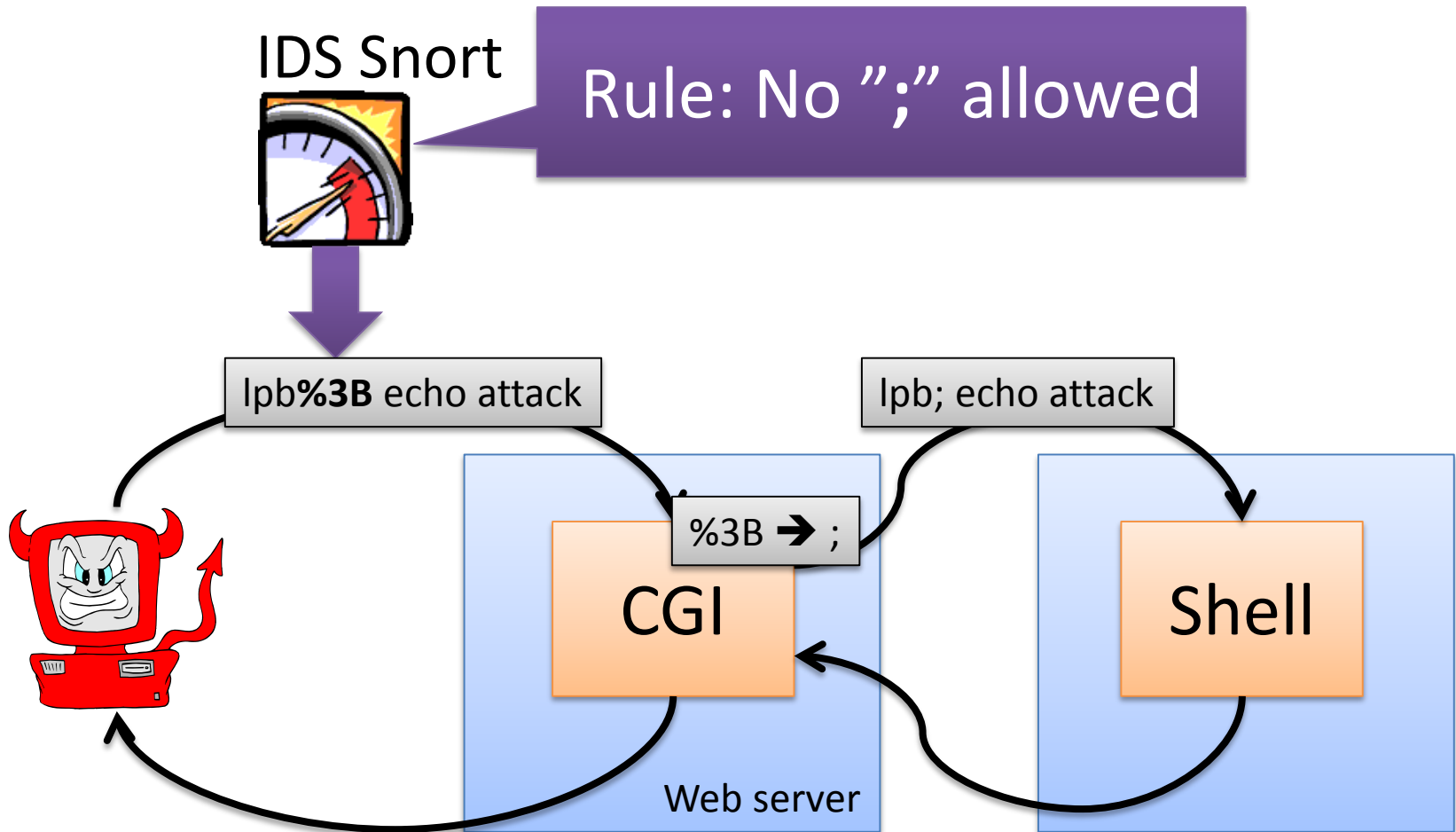
Example: Command Injection



Example: Command Injection



Example: Command Injection



OWASP

- The Open Web Application Security Project (OWASP) is ... organization focused on improving the security of application software.
- OWASP Local Chapter in Gothenburg
 - <https://www.owasp.org/index.php/Gothenburg>

OWASP Top 10 2013

- A1-Injection
- A2-Broken Authentication and Session Management
- A3-Cross Site Scripting (XSS)
- A4-Insecure Direct Object References
- A5-Security Misconfiguration
- A6-Sensitive Data Exposure
- A7-Missing Function Level Access Control
- A8- Cross Site Request Forgery (CSRF)
- A9-Using Known Vulnerable Components
- A10-Unvalidated Redirects and Forwards

OWASP Top 10 2013

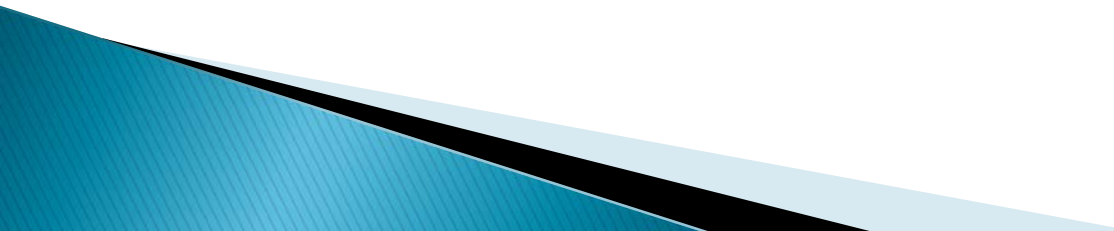
Name	Description
A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.



SQL injection example

Courtesy of John Smith, IBM Cooperation, GB

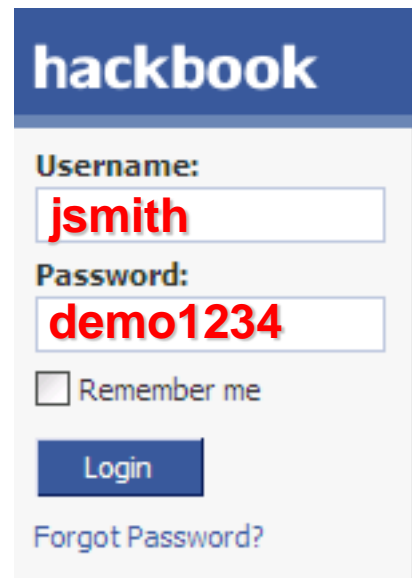
Injection Flaws

- ▶ What is it?
 - User-supplied data is sent to an interpreter as part of a command, query or data.
 - ▶ What are the implications?
 - SQL Injection – Access/modify data in DB
 - SSI Injection – Execute commands on server and access sensitive data
 - LDAP Injection – Bypass authentication
 - ...
- 

SQL Injection

User input is embedded as-is in predefined SQL statements:

```
query = "SELECT * from tUsers where  
userid='" + iUserID + "' AND  
password='" + iPassword + "'";
```



hackbook

Username:

Password:

Remember me

[Forgot Password?](#)



UserID	Username	Password	Name
1824	jsmith	demo1234	John Smith

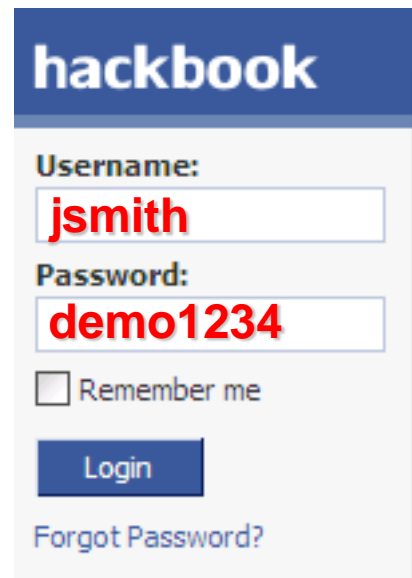
- Hacker supplies input that modifies the original SQL statement, for example: `iUserID = ' or 1=1 --`

```
SELECT * from tUsers where  
userid='' or 1=1 -- ' AND password='bar'
```

SQL Injection

User input is embedded as-is in predefined SQL statements:

```
query = "SELECT * from tUsers where  
userid='" + iUserID + "' AND  
password='" + iPassword + "'";
```



hackbook

Username:

Password:

Remember me

[Forgot Password?](#)



UserID	Username	Password	Name
1824	jsmith	demo1234	John Smith

- Hacker supplies input that modifies the original SQL statement, for example: `iUserID = ' or 1=1 --`



UserID	Username	Password	Name
1	admin	\$#kaoeFor	Admin

Summary

- ▶ Common theme with web application vulnerabilities:
 - Unvalidated user input is the attack vector
- ▶ Good security practice:
 - Assume all user input is evil !

OWASP Top 10 2013

Name	Description
A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

Google Gruyere

Web Application Exploits and Defenses

- Want to beat the hackers at their own game?
 - Learn how hackers find security vulnerabilities!
 - Learn how hackers exploit web applications!
 - Learn how to stop them!
- <http://google-gruyere.appspot.com/.../>
- Example
 - XSS attack: same origin policy
 - Injected code stored at site – run when user visits site (or tricking user to click on URL in email)
 - Here is an image of a cute
`<a href="http://google-gruyere.appspot.com/.../
<script>alert(1)</script>" >cat`



Web Application Exploits and Defenses

A Codelab by Bruce Leban, Mugdha Bendre, and Parisa Tabriz

UPDATED July 13, 2010: We have changed the name of the codelab application to Gruyere and have moved the location to this page. Please update your bookmarks.

Want to beat the hackers at their own game?

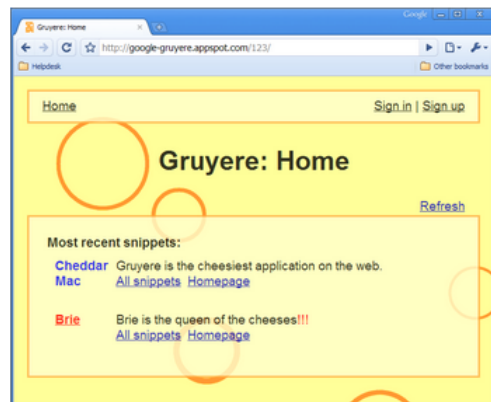
- Learn how hackers find security vulnerabilities!
- Learn how hackers exploit web applications!
- Learn how to stop them!

This codelab shows how web application vulnerabilities can be exploited and how to defend against these attacks. The best way to learn things is by doing, so you'll get a chance to do some real penetration testing, actually exploiting a real application. Specifically, you'll learn the following:

- How an application can be attacked using common web security vulnerabilities, like cross-site scripting vulnerabilities (XSS) and cross-site request forgery (CSRF).
- How to find, fix, and avoid these common vulnerabilities and other bugs that have a security impact, such as denial-of-service, information disclosure, or remote code execution.

To get the most out of this lab, you should have some familiarity with how a web application works (e.g., general knowledge of HTML, templates, cookies, AJAX, etc.).




Gruyere



This codelab is built around **Gruyere** /gru:'jɛər/ - a small, cheesy web application that allows its users to publish snippets of text and store assorted files. "Unfortunately," Gruyere has multiple security bugs ranging from cross-site scripting and cross-site request forgery, to information disclosure, denial of service, and remote code execution. The goal of this codelab is to guide you through discovering some of these bugs and learning ways to fix them both in Gruyere and in general.

The codelab is organized by types of vulnerabilities. In each section, you'll find a brief description of a vulnerability and a task to find an instance of that vulnerability in Gruyere. Your job is to play the role of a malicious hacker and find and exploit the security bugs. In this codelab, you'll use both black-box hacking and white-box hacking. In **black box hacking**, you try to find security bugs by experimenting with the application and manipulating input fields and URL parameters, trying to cause application errors, and looking at the HTTP requests and responses to guess server behavior. You do not have access to the source code, although understanding how to view source and being able to view http headers (as you can in Chrome or LiveHTTPHeaders for Firefox) is valuable. Using a web proxy like [Burp](#) or [WebScarab](#) may be helpful in creating or modifying requests. In **white-box hacking**, you have access to the source code and can use automated or manual analysis to identify bugs. You can treat Gruyere as if it's open source: you can read through the source code to try to find bugs. Gruyere is written in Python, so some familiarity with Python can be helpful. However, the security vulnerabilities covered are not Python-specific and you can do most of the lab without even looking at the code. You can run a local instance of Gruyere to assist in your hacking: for example, you can create an administrator account on your local instance to learn how administrative features work and then apply that knowledge to the instance you want to hack. Security researchers use both hacking techniques, often in combination, in real life.

We'll tag each challenge to indicate which techniques are required to solve them:

-  Challenges that can be solved just by using black box techniques.
-  Challenges that require that you look at the Gruyere source code.
-  Challenges that require some specific knowledge of Gruyere that will be given in the first hint.


WARNING: Accessing or attacking a computer system without authorization is illegal in many jurisdictions. While doing this codelab, you are specifically granted authorization to attack the Gruyere application as directed. You may not attack Gruyere in ways other than described in this codelab, nor may you attack App Engine directly or any other Google service. You should use what you learn from the codelab to make your own applications more secure. You should not use it to attack any applications other than your own, and only do that with permission from the appropriate authorities (e.g., your company's security team).

Gruyere: Home

[Refresh](#)

Most recent snippets:

Cheddar Mac Gruyere is the cheesiest application on the web.
[All snippets](#) [Homepage](#)

 **attacker** Do you know why garbage trucks drive so fast in Sweden? They are afraid of getting robbed...
[All snippets](#) [Homepage](#)

teacher All exam questions have been created!
[All snippets](#) [Homepage](#)

Brie Brie is the queen of the cheeses!!!
[All snippets](#) [Homepage](#)

Gruyere: Login

User name:

Password:

Gruyere: New Snippet

Add a new snippet.

```
Here is an image of a cute <a href="http://google-gruyere.appspot.com/400034139697/<script>alert(1)</script>">cat</a>
```

Limited HTML is now supported in snippets (e.g., , <i>, etc.)!

Submit

My Snippets

All snippets:

- 1 Here is an image of a cute [cat](#)
- 2 Do you know why garbage trucks drive so fast in Sweden? They are afraid of getting robbed

[My site](#)

[Home](#)

[Sign in](#) | [S](#)

Gruyere: Login

User name:


Password:

Gruyere: Home

[Refresh](#)

Most recent snippets:

Cheddar Mac Gruyere is the cheesiest application on the web.
[All snippets](#) [Homepage](#)

 **attacker** Here is an image of a cute [cat](#)
[All snippets](#) [Homepage](#)

teacher All exam questions have been created!
[All snippets](#) [Homepage](#)

Brie Brie is the queen of the cheeses!!!
[All snippets](#) [Homepage](#)

Invalid request: /

