Secure Software Development
Securing Systems

Objectives

- Explain how SQL injection occurs for an SQL statement
- Explain OS Command Injection
- Explain why scrubbing of memory helps to secure a system.
- Draw a picture explaining how cross site scripting occurs.
- Explain a man in the middle attack
- Explain how comments may compromise security
- Explain how security misconfiguration can impact system security
- Define blacklist and whitelist
- Explain techniques that can be used to protect memory.
- List the secure code characteristics

Injection Flaws

```
String sSQLQuery = "SELECT * FROM USERS
    WHERE user_id = '" + txtUSERID.Text + "' AND
    user_password = '" + txtPassword.Text + "'
```

OS Command Injection

- http://www.mycompany/com/sensitive/cgi-bin/userData.pl?doc=%20%3B%20/bin/ls%20-l%20

C++ STL <<

```
#include <iostream>

int main(int argc, char *argv[])
{
  char szTemp1[] = "After";
  char szTemp[16];
  char szTemp2[] = "Before";
  std::cin >> szTemp;
  std::cout << szTemp1 <<"\n";
  std::cout << "Input string " << szTemp<<"\n";
  std::cout << szTemp2<<"\n";
}
```

overflow

Character Arrays

Chara arrays? Use String

MSOE

C++ STL >> A better way

```
#include <iostream>

int main(int argc, char *argv[])
{
  char szTemp1[] = "After";
  char szTemp[16];
  char szTemp2[] = "Before";
  std::cin.width(16);
  std::cin >> szTemp;
  std::cout << szTemp1 <<"\n";
  std::cout << "Input string " << szTemp<<"\n";
  std::cout << szTemp2<<"\n";
}
```

C++ STL >> Even better yet

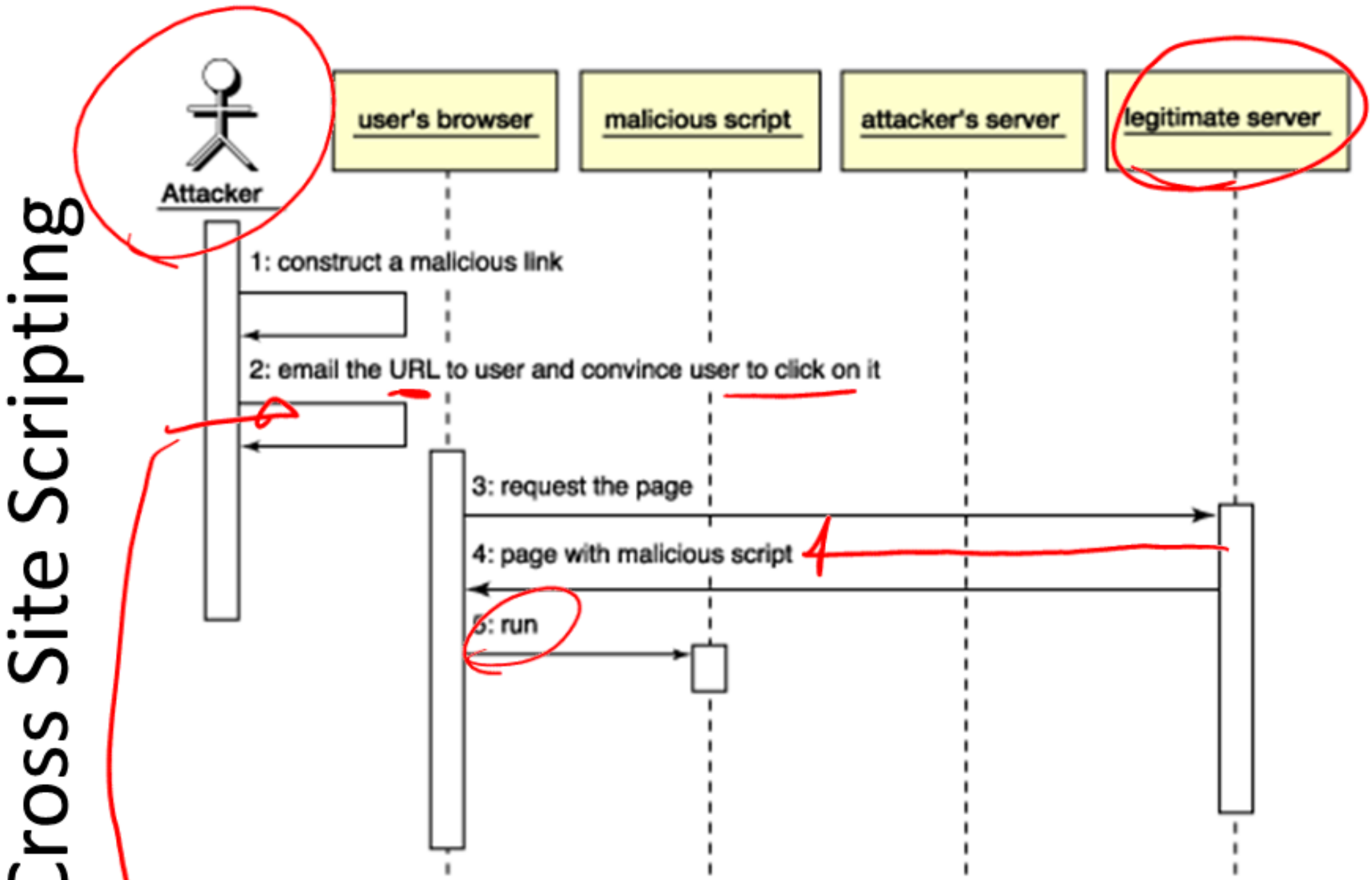
```
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char *argv[])
{
  string szTemp1 = "After";
  string szTemp;
  string szTemp2 = "Before";

  cin >> szTemp;
  cout << szTemp1 <<"\n";
  cout << "Input string " << szTemp<<"\n";
  cout << szTemp2<<"\n";
}
```

Scrubbing Memory

- [...]
  memset(key, 0, 32);
  [...]
- Fixed
  - [...]

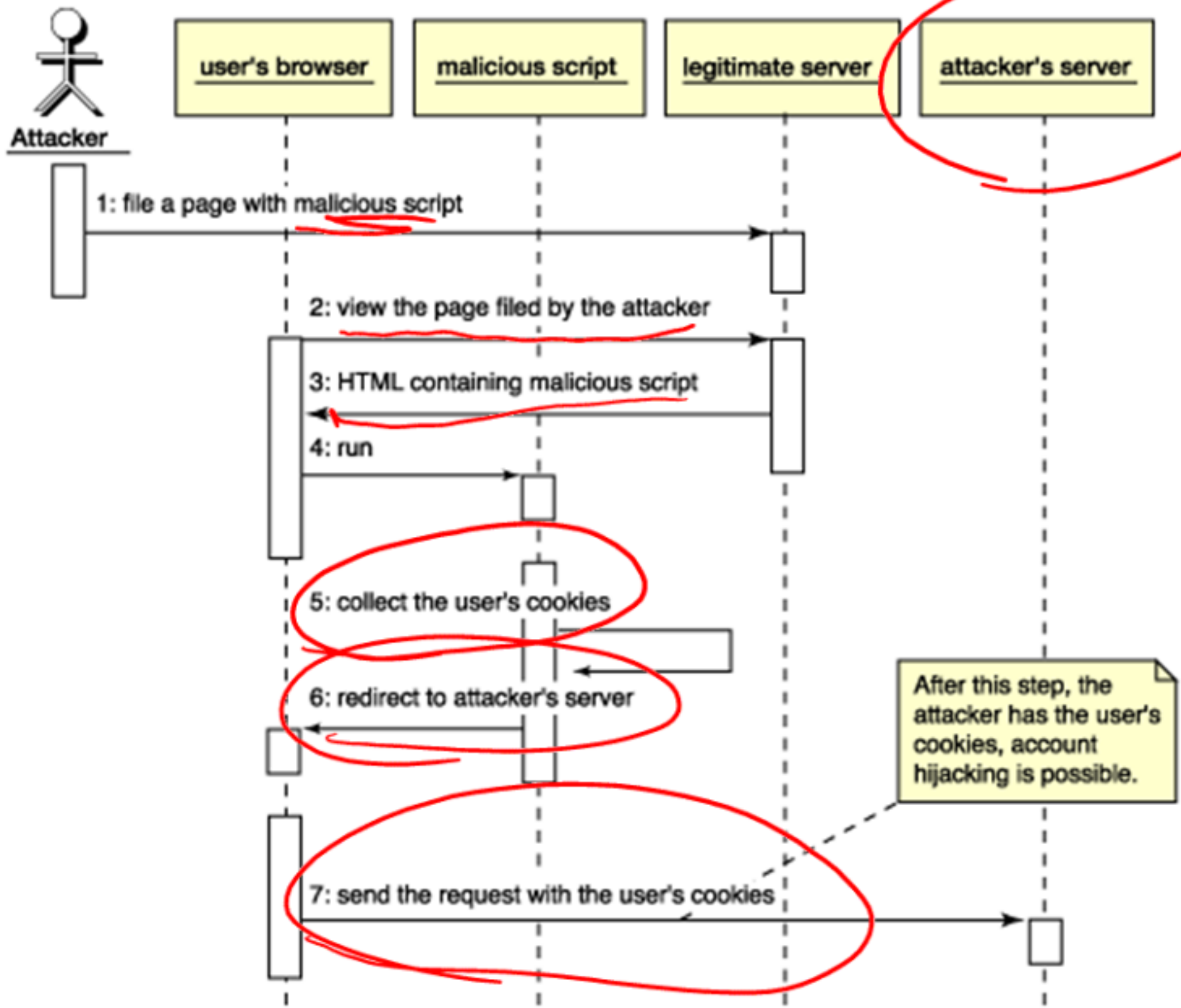
```
void *secureMemset(void *v,int c,size_t n){
volatile char *p = v;
while (n--)
  *p++ = c;
return v;
}
[...]
secureMemset(key, 0, 32);
[...]
```

Cross Site Scripting

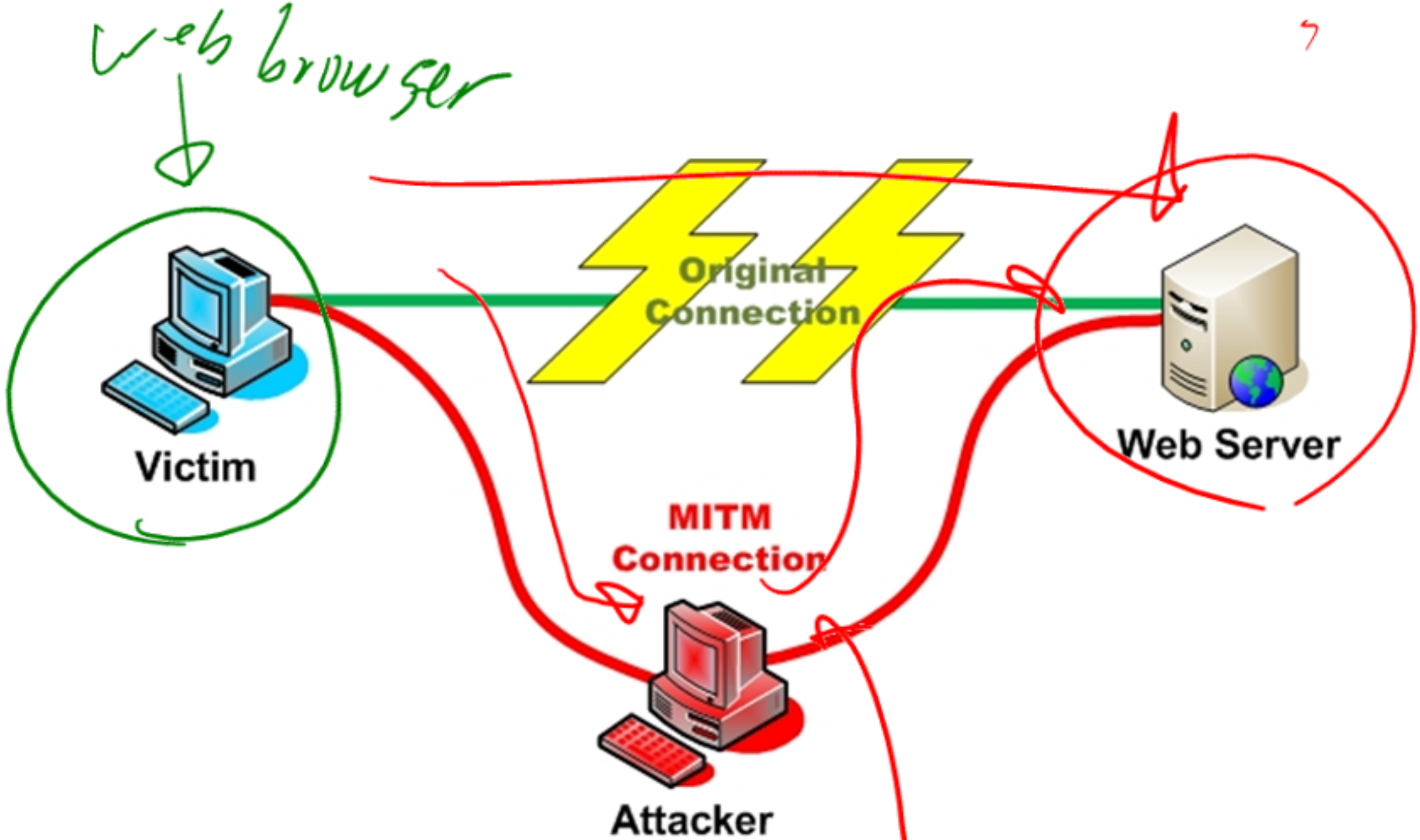


bank of amerila.ripoff.nv.ru

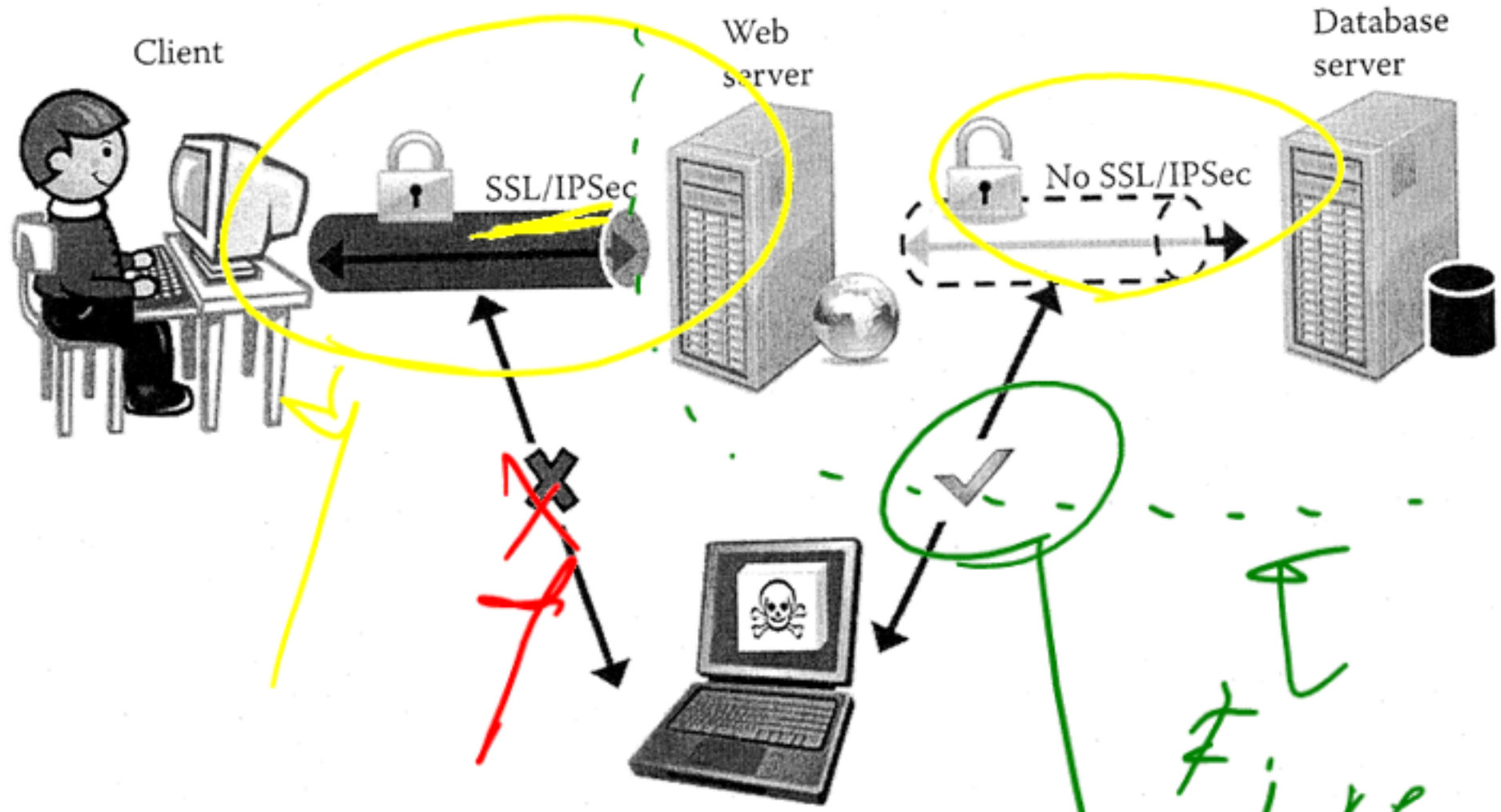
Cross Site Scriting



Man in the Middle Attack



Secure All Links



Not Likely

Defense in depth

Firewall

Security Hole

Security Related Comments in code...

```
/// <summary>
/// Establishes the connection to the database
/// </summary>
/// <param name="p_sServerName">Name of the database server.
/// Use HAMMERHEAD for Test and GREATWHITE for Production.
/// </param>
/// <param name="p_sLoginAccount">Name of the database login account.
/// Use Sally for Test and McQueen for Production.
/// </param>
/// <param name="p_sPassword">The password for the database login account.
/// Use Doc for Test and Mater for Production.
/// </param>
/// <param name="p_sDatabaseName">Name of the database to connect to.
/// Use PIXAR for Test and DREAMWORKS for Production.
/// </param>
private void BuildConnection(string p_sServerName,
    string p_sLoginAccount,
    string p_sPassword,
    string p_sDatabaseName)
{
    StringBuilder _oSBConnection = new StringBuilder();
    _oSBConnection.Append("Server=" + p_sServerName + ";");
    _oSBConnection.Append("uid=" + p_sLoginAccount + ";");
    _oSBConnection.Append("pwd=" + p_sPassword + ";");
    _oSBConnection.Append("Database=" + p_sDatabaseName + ";");

    SqlConnection _oSqlConn = new SqlConnection(_oSBConnection.ToString());
    _oSqlConn.Open();
}
```

Security Misconfiguration

- Hard Coded Credentials
- Not disabling the listing of directories or files on a web server
- Using default settings at installation
- Missing software patches
- Lack of perimeter controls

BAD

Fixing Vulnerabilities

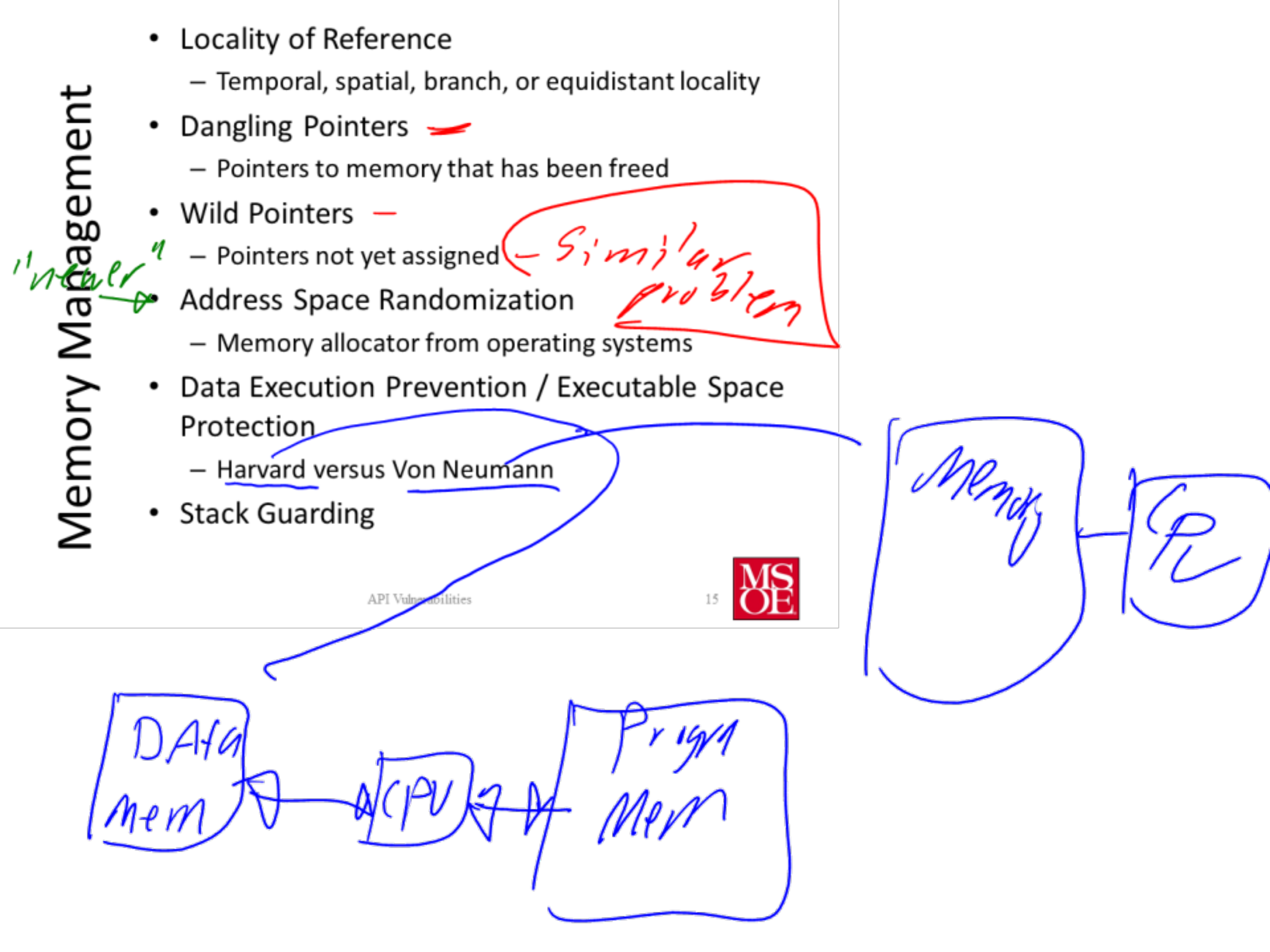
→ IT Administrator

Defensive Coding Techniques

- Input validation
  - Regular Expressions
- Filtration
  - Whitelist
    - Valid and non-malicious characters allowed in a string
  - Blacklist
    - Characters which are not allowed in an input field

Memory Management

- Locality of Reference
  - Temporal, spatial, branch, or equidistant locality
- Dangling Pointers
  - Pointers to memory that has been freed
- Wild Pointers
  - Pointers not yet assigned
- Address Space Randomization
  - Memory allocator from operating systems
- Data Execution Prevention / Executable Space Protection
  - Harvard versus Von Neumann
- Stack Guarding



Secure code characteristics

Source code characteristics

- Validates input.
- Does not allow dynamic construction of queries using user-supplied data.
- Audits and logs business-critical functions.
- Is signed to verify the authenticity of its origin.
- Does not use predictable session identifiers.
- Does not hard-code secrets inline.
- Does not cache credentials.
- Is properly instrumented.
- Handles exceptions explicitly.
- Does not disclose too much information in its errors.
- Does not reinvent existing functionality, and uses proven cryptographic algorithms.
- Does not use weak cryptographic algorithms.
- Uses randomness in the derivation of cryptographic keys.
- Stores cryptographic keys securely.
- Does not use banned APIs and unsafe functions.
- Is obfuscated or shrouded.
- Is built to run with least privilege.