IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 1, Ver. IV (Jan. 2014), PP 25-39
www.iosrjournals.org

# The Native Java Programming Language has suffered a Continuing Series of Vulnerabilities during the Past Two or More Years that Threaten to Destabilize Our Internet Technology.

Audu Jonathan Adoga, Lilian Nwanyanwu, Garba M. Rabiu
*Department of Computer Science, School of Science and Technology/Nasarawa State Polytechnic, Lafia, Nigeria*
*Department of Computer Science, School of Science and Technology/Nasarawa State Polytechnic, Lafia, Nigeria*
*Department of Computer Science, School of Science and Technology/Nasarawa State Polytechnic, Lafia, Nigeria*

***Abstract:*** *The native Java Programming Language whose importance to the internet cannot be over emphasised, has suffered series of vulnerabilities that threaten to destabilize the Internet Technology. Most of the vulnerabilities are introduced into the internet when the functionalities of a language are increased or when new version of a language is released but the discovery of these security holes and the provision of patches to these holes are usually done by either relevant professional organization such as OWASP and CVE or the inventor (s) of the language. This review tries to explain the concept of vulnerabilities, the native Java flaws and practical solutions to the vulnerabilities as proffered by relevant computer security organization(s).This will serve as a guide to software developers considering the devastating effects of the ignorance of secure coding methods.*
**Keywords:** *Flaws, Holes, Obfuscation, Patches, Vulnerability*

## I. Introduction

This report tries to support the fact that the native Java language has a lot of flaws, providing holes for possible attacks through fraudulent activities. Java exposures in the past years, have affected the stability of the internet. This is the cause of security awareness being created by security organizations about the current menace that has bedevilled the internet to the users.

Computer Security is an area that must be taken seriously so as to achieve high level of security despite the fact that the internet cannot be 100% secured .This is because, as new security holes are frequently discovered and patched, the attackers are relentless in their efforts to discover new means of gaining access to illegal information.

## II. Aims And Objectives

The aims and objectives of this review are as follows:

a To understand the concept of software vulnerabilities.
b To get information from secondary sources about Java vulnerabilities and how they affect the internet and to get professional solutions to some of the vulnerabilities from relevant authorities.
c At the end of this report, the concept of software vulnerability should be well understood. Some categories of Java flaws that affect the internet must have been well explained.
d The review promises to proffer solutions to some java flaws and they will serve as reference points to the relevant professionals so as to guard against bad coding practices.

## III. Literature Review

The term vulnerable is an adjective that means “exposed to being attacked or harmed” [1] while vulnerability is a noun form of the word vulnerable. Internet attacks have evolved over the years and this has raised serious concern about Internet Security. The main objectives of the attackers are to attack either vulnerable web servers or applications hosted by the web servers [2].

Moreover, a research conducted by Kuzma [3] shows that the use of digital libraries have witness great growth but the technical aspect of the service such as the protection of data, keeping the personal computers secure from malware has become a problem . This is because, most of the workers are not aware of security issues that can adversely affect their systems. In some cases, they workers may be careless about sensitive information.

Furthermore, the attackers have also shifted their attack focus towards the data of the end user because attackers know that, end users don’t have much knowledge about system security. The theft of data that are personal such as contact address, credit card numbers and other social numbers that are meant for security could be used for impersonation [2].

Several security flaws in Java have been discovered over the years despite the fact that Java has gained wide acceptance in the use of internet. One of the most recognised organizations that are concerned with the issue of software security awareness for users and professionals is the Open Web Application Security Project (OWASP). OWASP is a non-profit organization that helps people to make professional decision about software security risk. OWASP's Java Project has categorised security into so many subsections such as Session Management, Xpath Injection Java, Protection of Binaries, Miscellaneous Attacks, Input Validation, Error Handling and Logging, Cross-origin resource sharing, Content Security Policy and Authentication (https://www.owasp.org/index.php).

## IV. The Native Java Flaws And Solutions

This section tries to explain various types of Java flaws that exist and possible solutions to such flaws as proffered by relevant authorities. Most of the native vulnerabilities have been discussed by Open Web Application security Project (OWASP) Java project and other related organizations and this discussion is based on that.

### 4.1 Authentication

OWASP Java Project conducted by Prunet [4] showed in a project titled "Hashing java", that the Java applications we use on the internet mostly use login/password and they are stored as clear text in the database. This makes password vulnerable to injection attacks.

However, a solution to this problem was proffered with practical demonstration of what supposed to be done. Prunet, showed three steps using Java codes that could prevent authentication problem (check Appendix A: Hashing Java).

The first was to use one way encryption functions such as SHA-256, SHA-1 or MD5 that is good. According Prunet, an acceptable cryptographic function must produce a unique fingerprint (message digest) for every unique input.

Secondly, Prunet implemented an additional security code to the encryption by using what he called "salt" which is a randomly selected and fixed length number that was to be stored as plain text next to the password that has been hashed. The project also recommended that the iteration should be done n times as this could delay access time This may not affect the user much but affects an attacker since an attacker could spend long time trying to inject the system.

### 4.2 Input Validation

Java's HttpServletRequest was vulnerable [5]. An OWAP's release of Williams's project showed that it was possible to use getHeader, getCookie, getParameter methods to get information directly from user's web browsers due to HttpServletRequest vulnerability. Williams saw the need to build a class that is meant for validation into HttpServletRequest Object so that it becomes compulsory for a software developer to use this class during validation (check appendix B: How to Add Validation Logic to HttpServletRequest).

The project used HttpServletRequestWrapper together with JavaEE filter to control request. The newly developed class could override the HttpSevletRequest flaw by replacing them with series of calls that are meant for validation before anything could be returned. The customised method developed was called "validate". It could be called by validatingHttpRequest to throw exceptions that are related to validation if something is abnormal. Input was encoded before it was sent back to the application.

Furthermore, for any application developer to use the classes developed, it was recommended that the classes should be included in the class path of the application. This is to make all requests pass via the filter so that the throwing of exceptions and validation could be handed accurately.

### 4.3 Session Management

For security reasons, some variable called sessions are used to store information about a particular user. It is the host server that provides this storage. The part that is stored in a browser is called cookie. This is a unique way of identifying a user [6]. According to Nixon, moving from a web page to another web page can be tracked using session. Session can also be used to control the number of time a user is allowed to try logging with a wrong password, after which the system logs off the user for security reason. This is important but could be dangerous in Java if it is not implemented in the right way.

Righetto [7] in an OWASP's Java Project titled "logout", made it clear that session logout has the main function of terminating communication link between the browser and the web server.

Moreover, Righetto implemented two steps that could be used for a "global clean state". This prevents an intruder from using such identity. The implementation was done using java codes (appendix C: Logout) in two steps:

a Tell the host server that the session is no longer in use.

b Destroy all cookies sent by the host server to the browser that was used to track users.

### 4.4 Miscellaneous Attacks

In an OWASP Java Project carried out by Bergman [8] titled “Command Injection in Java”, Bergman made us to understand that there is a vulnerability in java when a developer tries to use system specific commands such as calling ls or dir from the shell. A developer was encouraged to use list function and a class in Java called Java File. The research further affirmed that developers should get used to java application programming interfaces instead of calling shell through the use of Runtime.exec file.

Bergman confirmed the proposition he made by developing two set of codes (appendix D: Command Injection in Java).The first set showed that, it was not possible to inject more commands if a particular user does not have full power over the arguments even though the user is allowed to take charge of the arguments that are supplied to the find command of windows. The second set of codes showed that it was possible for an attacker to have access to the system by using injection. This means, it was possible to carry our non-executable command with the help of user input through shell invoking.

### 4.5 Protecting Binaries

For a Java source code developed by a software developer to work in a different machine, it must be converted to a platform independent code called bytecode. Sometimes an application could be a very confidential type and the developer may not want reverse engineering of the bytecode. That is, converting the bytecode back to the source code. There exist a lot of tools on the internet that can decompile the bytecode thereby making the confidential application vulnerable [9].

Parrent, in an OWASP Java Project saw that a lot of tools that an attacker can use such as JAD, Jode and jReversePro exist. These tools were used by attackers to get JavaDoc and comments from a bytecode, or get source codes from the bytecode or get parameters and variable from the bytecode. It was a serious problem to confidential bytecode.

However, Parrend created awareness on various ways that could be used to prevent Java bytecode from being reversed. The research focus was further narrowed to a preventive method called obfuscation with a practical demonstration (appendix E: Bytecode Obfuscation). Obfuscation was achieved through variable renaming. The researcher also named various tools that could be used for obfuscation. Some of the tools mentioned were klassMaster, Proguard, Jarg, Javaguard and CafeBabe. The result is that, reverse engineering of confidential bytecode becomes very difficult.

### 4.6 Xpath Injection of Java

Based on recent research carried out on Xpath Injection [10] showed that it was possible to inject an application that used xml and JavaEE6 Servlet using Xpath Language. This could change the normal behaviour of a program so that, more information than expected could be retrieved from an xml data store or to retrieve different data from the expected.

Furthermore, the application could construct Xpath query that could select from stored data that was for employees. XML was used to store information about users. Two expressions for the selection of an employee’s data were written and a JavaEE6 Servlet code that was meant for selection was also written.

Due to the application’s vulnerability, the first expression assumed to be user input brought out information about all users in the employees’ store. The second expression was able to get information from the employees’ store by guessing. These were serious flaws.

However, Righetto was able to write two set of codes that could effectively handle the flaws. The first set of codes was able to check utility method and the second set of codes was able to handle utility use. Malicious Characters such as (, ), =, ‘, [, ], :, „, *, /, and white space that could cause problem to the application during input were effectively rejected by the codes that were written by Righetto (appendix F).

## V. Conclusion

With the clear, logical, rational and cutting edge research presented by renowned and relevant organizations such as the Open Web Security Application Project (OWASP), Network Security and Library Hi Tech in the area of system security, one could easily agree with the proposition that the native Java Programming Language has presented a lot of flaws in recent years that threaten the security of the internet.

However, the great efforts of OWASP and other related organizations towards securing the internet will transcend many centuries. OWASP’s offers of free awareness, free literary materials and free solutions to contemporary security problems are highly recognised. Software developers and other related professionals are advised to take advantage of the sample codes provided as appendixes which serve as guide during software development.

This report wishes to finally say this with great optimism that, though the malicious attackers keep advancing in their attacks of the internet, the forces of counter attacks shall be greater with great collaborations amongst security professionals and relevant organizations and the internet security of the future shall make web applications impenetrable to the malicious users.

## References

[1] Oxford, *oxfordenglish dictionary. 7thed*(Oxford: Oxford University Press,2012).

[2] Watson, D.,The Evolution Web Applications Attacks. *Network Security [e-journal] 2007(11)* Available through : Anglia Ruskin University Library website http://Libweb.anglia.ac.uk [Accessed 4 March 2013]

[3] Kuzma, J., European Digital Libraries: Web Security Vulnerabilities. *Library Hi Tech [e-journal] 28(3)*,2010 Available through : Anglia Ruskin University Library website http://Libweb.anglia.ac.uk [Accessed 4 March 2013]

[4] Prunet,M.,Hashing Java, *Open Web Application Security Project*,2008 [online] Available at< https://www.owasp.org/index.php/Hashing_Java> [Accessed 6 March, 2013]

[5] Williams, J., How to Add Validation Logic to HttpServletRequest, *Open Web Application Security Project,* 2008 [online] Available at<https://www.owasp.org/index.php/ How_to_add_validation_logic_to_HttpServletRequest> [Accessed 6 March, 2013]

[6] Nixon, R., *php, mysql, javascript&css*(Sebastopol: O'Reilly, Inc., 2012).

[7] Righetto,D.,Logout, *Open Web Application Security Project,*2012 [online] Available at<https://www.owasp.org/index.php/Logout> [Accessed 6 March, 2013]

[8] Bergman, N., Command Injection in Java*, Open Web Application Security Project,*2012 [online] Available at<https://www.owasp.org/index.php/command_injection_in_java> [Accessed 6 March, 2013]

[9] Parrend, P.,Bytecode Obfuscation*, Open Web Application Security Project*,2008 [online] Available at <https://www.owasp.org/index.php /Bytecode_obfuscation> [Accessed 6 March, 2013]

[10] Righetto, D.,Xpath Injection Java, *Open Web Application Security Project,*2012 [online] Available at< https://www.owasp.org/index.php/Xpath_Injection_Java> [Accessed 6 March, 2013]

**APPENDIX A: HASHING JAVA**
**COMPLETE JAVA SAMPLE**

In order to create the table needed by this application, call the method createTable(). It creates a TABLE called CREDENTIAL, with these fields :

- LOGIN VARCHAR (100) PRIMARY KEY
- PASSWORD VARCHAR (32)
- SALT VARCHAR (32)

In this database, the password and the salt are stored in Base64 representation.
The method *authenticate* is used in order to authenticate a user, the method *createUser* is used to create a new user.

```
packageorg.psafix.memopwd;

importjava.security.MessageDigest;
importjava.security.NoSuchAlgorithmException;
importjava.io.IOException;
import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;
importjava.sql.*;
importjava.util.Arrays;
importjava.security.SecureRandom;

public class Owasp {
private final static int ITERATION_NUMBER = 1000;

publicOwasp() {
   }

   /**
    * Authenticates the user with a given login and password
    * If password and/or login is null then always returns false.
    * If the user does not exist in the database returns false.
    * @param con Connection An open connection to a databse
    * @param login String The login of the user
    * @param password String The password of the user
    * @return boolean Returns true if the user is authenticated, false otherwise
    * @throws SQLExceptionIf the database is inconsistent or unavailable (
    *          (Two users with the same login, salt or digested password altered etc.)
    * @throws NoSuchAlgorithmException If the algorithm SHA-1 is not supported by the JVM
    */
publicboolean authenticate(Connection con, String login, String password)
throwsSQLException, NoSuchAlgorithmException{
boolean authenticated=false;
PreparedStatementps = null;
ResultSetrs = null;
try {
booleanuserExist = true;
          // INPUT VALIDATION
if (login==null||password==null){
             // TIME RESISTANT ATTACK
             // Computation time is equal to the time needed by a legitimate user
userExist = false;
login="";
password="";
          }

ps = con.prepareStatement("SELECT PASSWORD, SALT FROM CREDENTIAL WHERE LOGIN = ?");
ps.setString(1, login);
rs = ps.executeQuery();
          String digest, salt;
```

```
if (rs.next()) {
digest = rs.getString("PASSWORD");
salt = rs.getString("SALT");
            // DATABASE VALIDATION
if (digest == null || salt == null) {
throw new SQLException("Database inconsistant Salt or Digested Password altered");
            }
if (rs.next()) { // Should not append, because login is the primary key
throw new SQLException("Database inconsistent two CREDENTIALS with the same LOGIN");
            }
          } else { // TIME RESISTANT ATTACK (Even if the user does not exist the
            // Computation time is equal to the time needed for a legitimate user
digest = "000000000000000000000000000=";
salt = "00000000000=";
userExist = false;
          }

byte[] bDigest = base64ToByte(digest);
byte[] bSalt = base64ToByte(salt);

          // Compute the new DIGEST
byte[] proposedDigest = getHash(ITERATION_NUMBER, password, bSalt);

returnArrays.equals(proposedDigest, bDigest) &&userExist;
      } catch (IOException ex){
throw new SQLException("Database inconsistant Salt or Digested Password altered");
      }
finally{
close(rs);
close(ps);
      }
  }
  /**
   * Inserts a new user in the database
   * @param con Connection An open connection to a databse
   * @param login String The login of the user
   * @param password String The password of the user
   * @return boolean Returns true if the login and password are ok (not null and length(login)<=100
   * @throws SQLExceptionIf the database is unavailable
   * @throws NoSuchAlgorithmException If the algorithm SHA-1 or the SecureRandom is not supported by the
JVM
   */
publicbooleancreateUser(Connection con, String login, String password)
throwsSQLException, NoSuchAlgorithmException
  {
PreparedStatementps = null;
try {
if (login!=null&&password!=null&&login.length()<=100){
            // Uses a secure Random not a simple Random
SecureRandom random = SecureRandom.getInstance("SHA1PRNG");
            // Salt generation 64 bits long
byte[] bSalt = new byte[8];
random.nextBytes(bSalt);
            // Digest computation
byte[] bDigest = getHash(ITERATION_NUMBER,password,bSalt);
            String sDigest = byteToBase64(bDigest);
            String sSalt = byteToBase64(bSalt);

ps = con.prepareStatement("INSERT INTO CREDENTIAL (LOGIN, PASSWORD, SALT) VALUES (?,?,?)");
```

```
ps.setString(1,login);
ps.setString(2,sDigest);
ps.setString(3,sSalt);
ps.executeUpdate();
return true;
        } else {
return false;
        }
      } finally {
close(ps);
      }
   }

   /**
    * From a password, a number of iterations and a salt,
    * returns the corresponding digest
    * @paramiterationNbintThe number of iterations of the algorithm
    * @param password String The password to encrypt
    * @param salt byte[] The salt
    * @return byte[] The digested password
    * @throws NoSuchAlgorithmException If the algorithm doesn't exist
    */
public byte[] getHash(intiterationNb, String password, byte[] salt) throws NoSuchAlgorithmException {
MessageDigest digest = MessageDigest.getInstance("SHA-1");
digest.reset();
digest.update(salt);
byte[] input = digest.digest(password.getBytes("UTF-8"));
for (int i = 0; i <iterationNb; i++) {
digest.reset();
input = digest.digest(input);
      }
return input;
   }


public void createTable(Connection con) throws SQLException{
      Statement st = null;
try {
st = con.createStatement();
st.execute("CREATE TABLE CREDENTIAL (LOGIN VARCHAR(100) PRIMARY KEY, PASSWORD
VARCHAR(32) NOT NULL, SALT VARCHAR(32) NOT NULL)");
      } finally {
close(st);
      }
   }

   /**
    * Closes the current statement
    * @paramps Statement
    */
public void close(Statement ps) {
if (ps!=null){
try {
ps.close();
        } catch (SQLException ignore) {
        }
      }
   }
```

```
/**
 * Closes the current resultset
 * @paramps Statement
 */
public void close(ResultSetrs) {
if (rs!=null){
try {
rs.close();
        } catch (SQLException ignore) {
        }
    }
  }
  /**
   * From a base 64 representation, returns the corresponding byte[]
   * @param data String The base64 representation
   * @return byte[]
   * @throws IOException
   */
public static byte[] base64ToByte(String data) throws IOException {
    BASE64Decoder decoder = new BASE64Decoder();
returndecoder.decodeBuffer(data);
  }

  /**
   * From a byte[] returns a base 64 representation
   * @param data byte[]
   * @return String
   * @throws IOException
   */
public static String byteToBase64(byte[] data){
    BASE64Encoder endecoder = new BASE64Encoder();
returnendecoder.encode(data);
  }

 }
```

Source: Prunet, M.,(2008).Hashing Java,Open Web Application Security Project(OWASP). https://www.owasp.org/index.php/Hashing_Java

**APPENDIX B: HOW TO ADD VALIDATION LOGIC TO HTTPSERVLETREQUEST**

```
public class ValidatingHttpRequest extends HttpServletRequestWrapper {

public ValidatingHttpRequest(HttpServletRequest request) {
super(request);
    }

public String getParameter(String name) {
        HttpServletRequest req = (HttpServletRequest) super.getRequest();
return validate( name, req.getParameter( name ) );
    }

    // Danger - you can optionally allow getting the raw parameter
public String getRawParameter( String name ) {
        HttpServletRequest req = (HttpServletRequest) super.getRequest();
returnreq.getParameter( name );
    }
    ... follow this pattern for getHeader(), getCookie(), etc...
    ... specifically don´t forget getParameterValues() as this is used by frameworks like Struts to get the parameter values
```

... Struts2 uses getParameterMap() to get the parameter values. Below is the sample way how to validate that.

```
public Map<String, String[]>getParameterMap() {
Map<String, String[]> map = super.getParameterMap();
Iterator iterator = map.keySet().iterator();
                Map<String, String[]>newMap = new LinkedHashMap<String, String[]>();

        while (iterator.hasNext()) {
                String key = iterator.next().toString();

                String []values = map.get(key);
                String []newValues = new String[values.length];

        for(int i = 0; i <values.length; i++){
                newValues[i] = validate(key, values[i]); // Apply validation logic to the value
                }

        newMap.put(key, newValues);
          }
        returnnewMap;
    }
    // This is a VERY restrictive pattern alphanumeric < 20 chars
    // It's easy to make this a parameter for the filter and configure in web.xml
private Pattern pattern = Pattern.compile("^[a-zA-Z0-9]{0,20}$");

private String validate( String name, String input ) throws ValidationException {
        // important - always canonicalize before validating
        String canonical = canonicalize( input );

        // check to see if input matches whitelist character set
if ( !pattern.matcher( canonical ).matches() ) {
throw new ValidationException( "Improper format in " + name + " field";
        }

        // you could html entity encode input, but it's probably better to do this before output
        // canonical = HTMLEntityEncode( canonical );

return canonical;
    }

    // Simplifies input to its simplest form to make encoding tricks more difficult
private String canonicalize( String input ) {
        String canonical = sun.text.Normalizer.normalize( input, Normalizer.DECOMP, 0 );
return canonical;
    }
    //--A.in.the.k 08:57, 19 March 2009 (UTC) for correct implementation see
http://www.owasp.org/index.php/How_to_perform_HTML_entity_encoding_in_Java
    // Return HTML Entity code equivalents for any special characters
public static String HTMLEntityEncode( String input ) {
StringBuffersb = new StringBuffer();
for ( int i = 0; i <input.length(); ++i ) {
charch = input.charAt( i );
if ( ch>='a' &&ch<='z' || ch>='A' &&ch<='Z' || ch>='0' &&ch<='9' ) {
sb.append(ch );
            } else {
sb.append( "&#" + (int)ch + ";" );
            }
        }
returnsb.toString();
```

```
        }

    }
```

Then all we have to do is make sure that all the requests in our application get wrapped in our new wrapper. It's easy to implement with a Java EE filter.

```
public class ValidationFilter implements Filter {
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) {
chain.doFilter(new ValidatingHttpRequest( (HttpServletRequest)request ), response);
        }
    }
```

To add the filter to our application, all we have to do is put these classes on our application's classpath and then set up the filter in web.xml.

```
<filter>
<filter-name>ValidationFilter</filter-name>
<filter-class>ValidationFilter</filter-class>
</filter>

<filter-mapping>
<filter-name>ValidationFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
```

Source: Williams, J.,(2008).How to Add Validation Logic to HttpServletRequest, Open Web Application Security Project(OWASP).
**https://www.owasp.org/index.php/ How_to_add_validation_logic_to_HttpServletRequest**

**APPENDIX C: LOGOUT**
**CODE SAMPLE OF SESSION LOGOUT PROCESS**

```
packageorg.owasp.javaproject.logout;

importjava.io.IOException;
importjavax.servlet.ServletException;
importjavax.servlet.annotation.WebServlet;
importjavax.servlet.http.Cookie;
importjavax.servlet.http.HttpServlet;
importjavax.servlet.http.HttpServletRequest;
importjavax.servlet.http.HttpServletResponse;
importjavax.servlet.http.HttpSession;

/**
 * Code sample showing how to perform a complete logout
 */
@SuppressWarnings("serial")
@WebServlet("/Logout")
public class LogoutCodeSample extends HttpServlet {

        /**
         * {@inheritDoc}
         *
         * @see javax.servlet.http.HttpServlet#doPost(javax.servlet.http.HttpServletRequest,
         *      javax.servlet.http.HttpServletResponse)
         */
        @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                doGet(request, response);
        }
```

```
	/**
	 * {@inheritDoc}
	 *
	 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
	 *      response)
	 */
	@Override
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		/*
		 * First step : Invalidate user session
		 */
		HttpSession session = request.getSession(false);
		if (session != null) {
			session.invalidate();
		}

		/*
		 * Second step : Invalidate all cookies by, for each cookie received,
		 * overwriting value and instructing browser to deletes it
		 */
		Cookie[] cookies = request.getCookies();
		if (cookies != null &&cookies.length> 0) {
			for (Cookie cookie : cookies) {
				cookie.setValue("-");
				cookie.setMaxAge(0);
				response.addCookie(cookie);
			}
		}

	}

}
```

Source: Righetto,D.,(2012).Logout, Open Web Application Security Project(OWASP).
**https://www.owasp.org/index.php/Logout**

**APPENDIX D: COMMAND INJECTION IN JAVA**
**EXAMPLE 1**
The code below allows a user to control the arguments to the Window's *find* command. While the user does have full control over the arguments, it is not possible to inject additional commands. For example, inputting “test & del file” will not cause the *del* command to execute, since Runtime.exec tokenizes the command string and then invokes the *find* command using the parameters “test”, “&”, “del”, and “file.”

```
import java.io.*;

public class Example1 {
	public static void main(String[] args)
	throwsIOException {
		if(args.length != 1) {
			System.out.println("No arguments");
			System.exit(1);
		}
		Runtime runtime = Runtime.getRuntime();
		Process proc = runtime.exec("find" + " " + args[0]);

		InputStream is = proc.getInputStream();
		InputStreamReaderisr = new InputStreamReader(is);
		BufferedReaderbr = new BufferedReader(isr);
```

```
                String line;
                while ((line = br.readLine()) != null) {
                        System.out.println(line);
                }
        }
}
```

**EXAMPLE 2**

The code below invokes the system shell in order to execute a non-executable command using user input as parameters. Non-executable Window's commands such as *dir* and *copy* are part of the command interpreter and therefore cannot be directly invoked by Runtime.exec. In this case, command injection is possible and an attacker could chain multiple commands together. For example, inputting ". & echo hello" will cause the *dir* command to list the contents of the current directory and the *echo* command to print a friendly message.

```
import java.io.*;

public class Example2 {
        public static void main(String[] args)
        throwsIOException {
                if(args.length != 1) {
                        System.out.println("No arguments");
                        System.exit(1);
                }
                Runtime runtime = Runtime.getRuntime();
                String[] cmd = new String[3];
                cmd[0] = "cmd.exe" ;
cmd[1] = "/C";
cmd[2] = "dir " + args[0];
                Process proc = runtime.exec(cmd);

                InputStream is = proc.getInputStream();
                InputStreamReaderisr = new InputStreamReader(is);
                BufferedReaderbr = new BufferedReader(isr);

                String line;
                while ((line = br.readLine()) != null) {
                        System.out.println(line);
                }
        }
}
```

Source: Bergman,N.,(2012).Command Injection in Java, Open Web Application Security Project(OWASP).
**https://www.owasp.org/index.php/command_injection_in_java**

**APPENDIX E: BYTECODE OBFUSCATION**
**USING PROGUARD**

The following section provides a short tutorial for using Proguard.
First, download the code under following urland unzip it.
For this tutorial, we use the fr.inria.ares.sfelixutils-0.1.jar package.
Go to the main directory of Proguard. For lauching it, you can use following script with given parameters :

```
java -jar lib/proguard.jar @config-genericFrame.pro
```

config-genericFrame.pro is the option file (do not forget to adapt the libraryjars parameter to your own system) :

```
-obfuscationdictionary ./examples/dictionaries/compact.txt
-libraryjars /usr/java/j2sdk1.4.2_10/jre/lib/rt.jar
-injars fr.inria.ares.sfelixutils-0.1.jar
-outjar fr.inria.ares.sfelixutils-0.1-obs.jar
-dontshrink
-dontoptimize
-keep public class proguard.ProGuard {
public static void main(java.lang.String[]);
}
```

Remark that the 'keep' option is mandatory, we use this default class for not keep anything out.
The example dictionnary (here compact.txt) is given with the code.
The output is stored in the package 'genericFrameOut.jar'.
You can observe the modifications implied by obfuscation with following commands :
jarxvf genericFrameOut.jar
cdgenericFrame/pub/gui/
jadc.class
morec.jad more c.jad
Remark than Strings are kept unmodified. If you want you code to be hard to read, do not forget to remove any debugging and logging comments. Jode has some facilities for making this easier.

**USING CAFEBABE**

CafeBabe is a convenient tool for teaching structure of ByteCode files. You can download it at this URL.
Unzip it and execute following command :
java -classpath CafeBabe.jar org.javalobby.apps.cafebabe.CafeBabe
Have a look at some class from the original genericFrame.jar package.
Then obfuscate it, and compare both - original and modified class :

- with the CafeBabe viewer,
- after decompiling it with JAD.

What conclusion can you draw of it ?

**Using Jode**

Jode is to be found here with instructions on how to use the decompiler and obfuscator functions here.
Source: Parrend,P.,(2008).Bytecode Obfuscation, Open Web Application Security Project(OWASP).
**https://www.owasp.org/index.php /Bytecode_obfuscation**

**APPENDIX F: Xpath Injection Java**

**Injection examples**

For the examples we will take a case of an application that store employees informations using XML store with this structure:

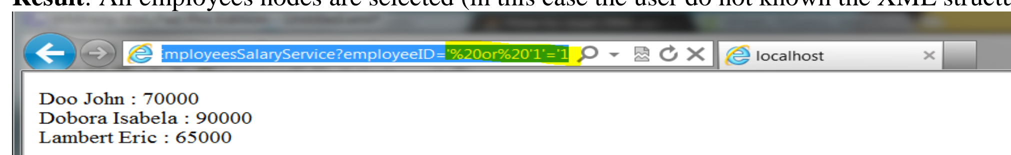
```
<?xml version="1.0" encoding="utf-8"?>
<employees>
<employee id="AS789" firstname="John" lastname="Doo" annualsalary="70000"/>
<employee id="AS719" firstname="Isabela" lastname="Dobora" annualsalary="90000"/>
<employee id="AS219" firstname="Eric" lastname="Lambert" annualsalary="65000"/>
</employees>
```

The XPATH expression to select an employee node used by application is:
/employees/employee[@id='EMPLOYEE_ID']
The code (JavaEE6 Servlet for example) used to perform selection is:

```
packageorg.owasp.javaproject.xpathinjection;

importjava.io.IOException;
importjava.io.StringReader;
importjava.util.List;

importjavax.servlet.ServletException;
importjavax.servlet.annotation.WebServlet;
importjavax.servlet.http.HttpServlet;
importjavax.servlet.http.HttpServletRequest;
importjavax.servlet.http.HttpServletResponse;
importjavax.xml.parsers.DocumentBuilder;
importjavax.xml.parsers.DocumentBuilderFactory;

importorg.jaxen.XPath;
importorg.jaxen.dom.DOMXPath;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
importorg.xml.sax.InputSource;

/**
 * Sample service to retrieve employees salary
```

```
 */
@SuppressWarnings("serial")
@WebServlet("/EmployeesSalaryService")
public class EmployeesSalaryService extends HttpServlet {
	private static final String DATASOURCE_XML = "Put XML Structure above here";
	/**
	 * {@inheritDoc}
	 *
	 * @see javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
	 */
	@SuppressWarnings("rawtypes")
	@Override
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		try {
			// For the sample we load the XML Document at each request but this not a good way
for real application.....
			DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
			DocumentBuilder builder = factory.newDocumentBuilder();
			Document doc = builder.parse(new InputSource(new
StringReader(DATASOURCE_XML)));
			// Retrieve employee ID from the input HTTP request
			String eID = request.getParameter("employeeID");
			if (eID == null) {
				eID = "";
			}
			// Create XPATH expression
			String xpathExpr = "/employees/employee[@id='" + eID + "']";
			XPath expression = new DOMXPath(xpathExpr);

			// Apply expression on XML document
			List nodes = expression.selectNodes(doc);
			for (int i = 0; i <nodes.size(); i++) {
				Element employee = (Element) nodes.get(i);
				response.getWriter().print(employee.getAttribute("lastname")
				+ " "
				+ employee.getAttribute("firstname")
				+ " : "
				+ employee.getAttribute("annualsalary")
				+ "<br>");
			}
		} catch (Exception e) {
			response.sendError(HttpServletResponse.SC_BAD_REQUEST);
			e.printStackTrace();
		}
	}
}
```

Here the sensitive information is the annual salary then it's will be the target of the injection.
The application expect to receive, for the employee ID, an value like "AS789" but what is the application behavior if a user submit another value pattern ?
Sample value n°1:
'%20or%20'1'='1
**Result**: All employees nodes are selected (in this case the user do not known the XML structure).



Sample value n°2:
'%20or%20fn:contains(fn:lower-case(@lastname),'dobora')%20or%20'
**Result**: Employee where the last name contains "dobora" is selected (in this case the user has guessed the XML structure).

**Injection countermeasure**
Input used into an XPATH expression must not contains any of the characters below:
( ) = ' [ ] : , * / WHITESPACE
According to our example context, the modification to apply could be to create an application transversal utility method checking the presence of characters above and rejecting the value submitted if it's contains any one.

Checking utility method example:

```
publicbooleancheckValueForXpathInjection(String value) throws Exception {
	booleanisValid = true;
	if ((value != null) && !"".equals(value)) {
		String xpathCharList = "()='[]:,*/ ";
		// Always to avoid encoding evading....
		String decodedValue = URLDecoder.decode(value, Charset.defaultCharset().name());
		for (char c : decodedValue.toCharArray()) {
			if (xpathCharList.indexOf(c) != -1) {
				isValid = false;
				break;
			}
		}
	}
	returnisValid;
}
```

Checking utility use example:

```
/**
 * {@inheritDoc}
 *
 *	@see	javax.servlet.http.HttpServlet#doGet(javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)
 */
@SuppressWarnings("rawtypes")
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
	try {
		// Check input
		if (!checkValueForXpathInjection(request.getParameter("employeeID"))) {
			response.sendError(HttpServletResponse.SC_BAD_REQUEST);
			// Trace injection
			// Exit
			return;
		}
	.....

		}
	} catch (Exception e) {
		response.sendError (HttpServletResponse.SC_BAD_REQUEST);
		e.printStackTrace();
	}
}
```

Source: Righetto D., (2012).Xpath Injection Java, Open Web Application Security Project(OWASP).
**https://www.owasp.org/index.php/Xpath_Injection_Java**