Randori is a training tool that generates randomized content for computer science instruction. Randori can generate vulnerable source code and a key describing the flaws in the file or files created.

Randori randomly chooses content from templates, and generates output files that contain that content. It also generates a key associated with that content from the template. This application can generate a user defined number of files and keys, making it suitable for use in the classroom.

Running Randori

The Randori application was written in Python 2.6.5 for Linux.

To run this application use the command:

```
python randori.py –i authN.xml –n 3
```

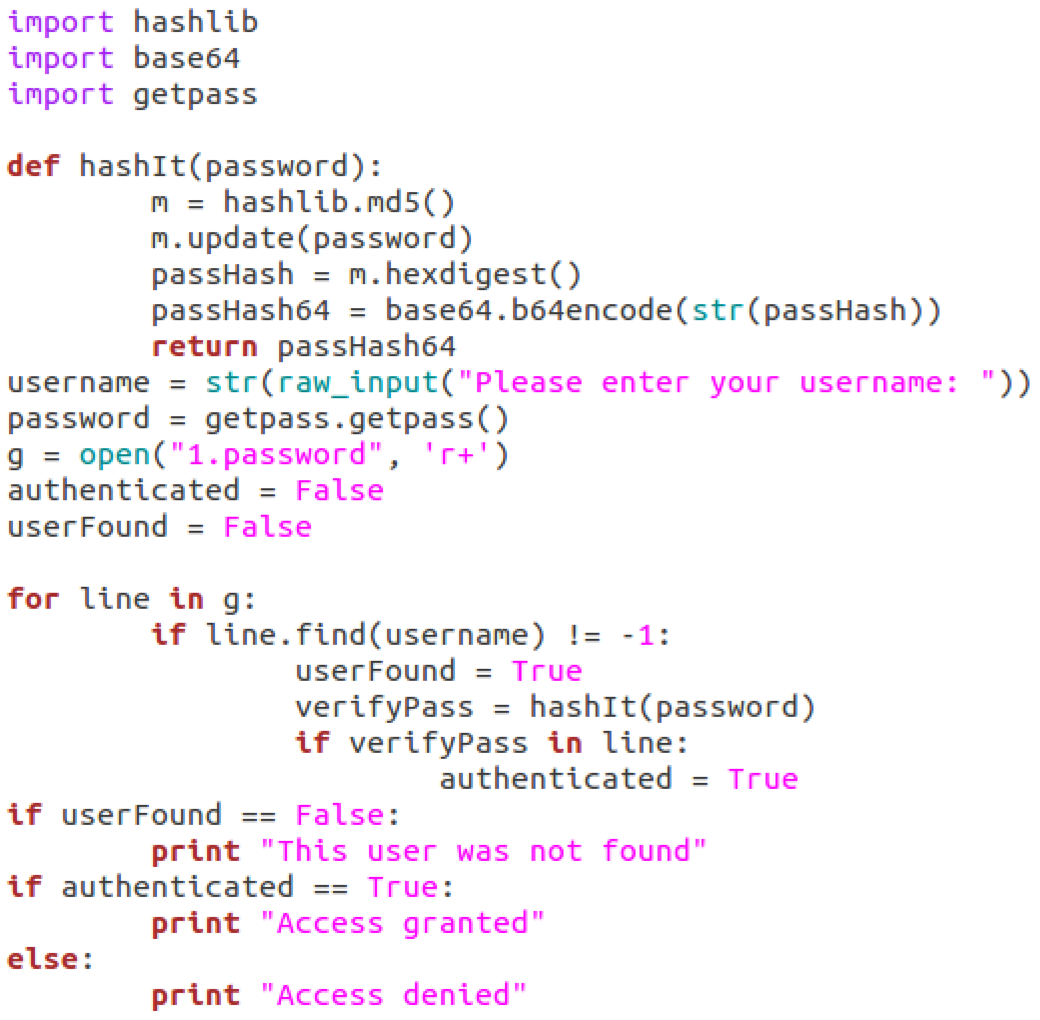
Enter the name of the xml template file you want to use to create files and the number of files you want created. In this case we have used the authN.xml template. This template will generate random python application source code with authentication vulnerabilities.

The files will be created with the file-type of the template preceded by an integer. The key files will be name “KEY” then the number of the file that they are the key to.

```
user$ python randori.py –i authN.xml –n 30
###############################################################
randori engine
###############################################################
The files have been created
```

In this example, the files created from the authN.xml template are a python file and a locally stored password file. These files will be numbered one through thirty. Each of these files will have a corresponding key file that details the security issues found in the application created.

The first file created is 1.py. Here is one possible example of a 1.py file that could be created:



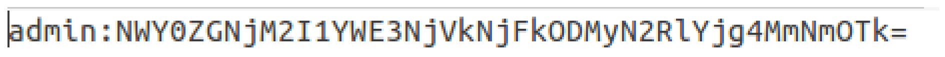
```
import hashlib
import base64
import getpass

def hashIt(password):
        m = hashlib.md5()
        m.update(password)
        passHash = m.hexdigest()
        passHash64 = base64.b64encode(str(passHash))
        return passHash64
username = str(raw_input("Please enter your username: "))
password = getpass.getpass()
g = open("1.password", 'r+')
authenticated = False
userFound = False

for line in g:
        if line.find(username) != -1:
                userFound = True
                verifyPass = hashIt(password)
                if verifyPass in line:
                        authenticated = True
if userFound == False:
        print "This user was not found"
if authenticated == True:
        print "Access granted"
else:
        print "Access denied"
```

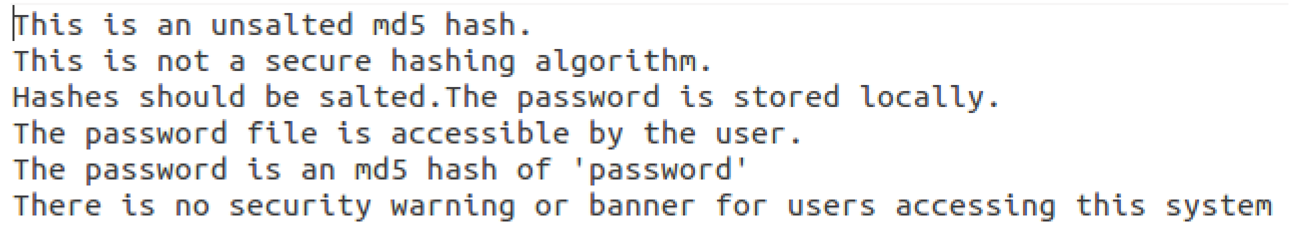
This vulnerable application requests a user-name and password from a user, hashes the password, compares it with the password in the password file, and states whether the access has been granted to the user.

The other file from this template is the corresponding password file for this application. For this python file, this file will be named 1.password and contain a user-name and hash of the password.



```
admin:NWY0ZGNjM2I1YWE3NjVkNjFkODMyN2RlYjg4MmNmOTk=
```

The key file will contain the security errors written to the file when it was created. This file will be named 1KEY.txt.



```
This is an unsalted md5 hash.
This is not a secure hashing algorithm.
Hashes should be salted.The password is stored locally.
The password file is accessible by the user.
The password is an md5 hash of 'password'
There is no security warning or banner for users accessing this system
```

After running this command there will be 30 sets of randomly created files and keys. The randomly created files can be provided to students to evaluate, and the key files can be kept by the instructor to evaluate the student assessment.

Sample Lesson Plan Using Randori

INSTRUCTIONAL GOAL

Students should be able to demonstrate the ability to recognize a vulnerable implementation of user authentication to an application.

They should be able to bypass the authentication of an application.

They should be able to identify the presence or absence of cryptographic hashing algorithms.

They should know to try common passwords when guessing passwords

PERFORMANCE OBJECTIVE

Students will identify common vulnerabilities in an application that authenticates users.

Students will identity whether passwords entered by users were masked.

Students will identify the presence or absence of cryptographic hashing.

Students will be able to demonstrate the ability to bypass a weak implementation of user authentication.

LESSON CONTENT

Password storage

Common weak passwords

Cryptographic hashing

Weak cryptographic hashing algorithms

INSTRUCTIONAL PROCEDURES

- The instructor will instruct the student in cryptographic hashing, common weak passwords, password storage, and weak cryptographic hashing algorithms.
- The instructor will run the randori engine and generate a python source code file, the associated password file, and a key file for each student.
- The students will be provided with the python source and associated password file. The instructor will use the key file to asses the students ability to identify the vulnerabilities present in the file.
- The student will be instructed to document how to bypass the authentication on the application and assess security vulnerabilities in the application.
- The student's assessment will be compared with the key and scored based on how many vulnerabilities were present and how many of those they were able to identify

Making your own templates

Randori Engine

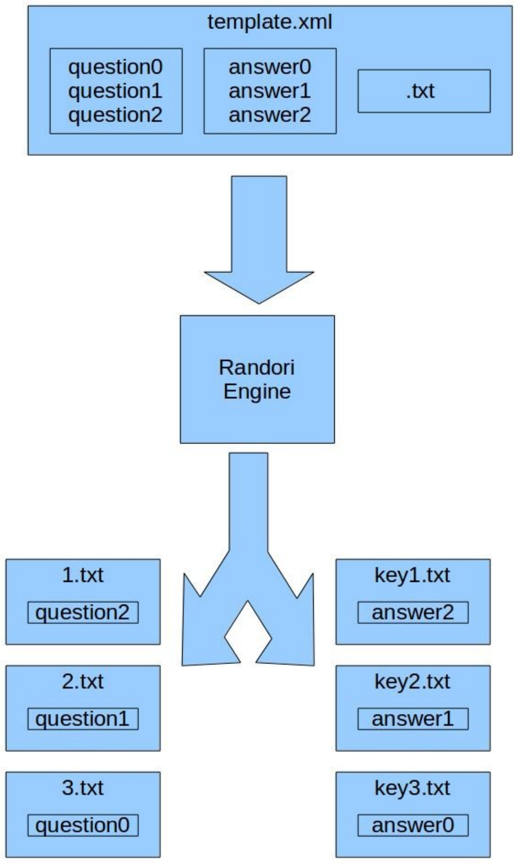
The Randori engine takes files written in XML and creates a randomized instance of the output described in the file and a key describing the random elements in the file. The Randori engine is written in Python and will only process properly formatted files written in valid XML.

Randori Mark Up Language

The Randori template files use a mark up language written in XML for creating random files with the Randori engine. It defines templates for randomly created files. The templates include the file type of the files to be created, the content of those files, and the key information related to those files. This key information is related to specific randomly chosen content present in the files created. This content could also be the answer to a question or problem defined in the content of the template.

Overview

The template contains the content and the file type for the file to be created. In this case the contents are three questions and answers. Here the content and answers defined in the XML template are numbered zero to three. The file type is defined to be txt.



The template is read into the Randori engine. Here the user has asked for three files to be made from the template.xml file.

The Randori engine will generate three sequentially numbered .txt files containing randomly chosen content. It will also generate three key files containing the correct answers to those questions.

Randori Mark Up Language

A Randori mark up language file is an XML mark up language that defines documents that can be created using the Randori engine. The type of file or files to be made and the content of those files are defined in the file. The file type is tagged in the types tag. The content of the output can be static or dynamic. This content is tagged as either static or dynamic.

Types

The file type of the file or files that will contain the content is defined in the types tag.

Here is what would be used to create a text file of type .txt:

```
<types><a>txt</a></types>
```

The Randori engine will create incrementally numbered files with this file type. If a user chose to make two files from this template, the Randori engine would create files named 1.txt and 2.txt.

Here is what would be used to create two different files of different file types:

```
<types><a>h</a><b>cpp</b></types>
```

The Randori engine will create two incrementally numbered files with these file types. If a user chose to make two files from this template, the Randori engine would create files named 1.h, 1.cpp, 2.h, and 2.cpp.

Here is what should be used to create two different files of the same file type:

```
<types><a>py</a><b>other.py</b></types>
```

The Randori engine will create two incrementally numbered files with the file type of .py. To make sure that there are two separate files, additional text needs to be added to the string that will be used to make the other .py file. If a user chose to make two files from this template, the Randori engine would create files named 1.py, 1.other.py, 2.py, and 2.other.py.

Dynamic

The dynamic tag is used to define dynamic content. Dynamic content is randomly chosen content to be included in the output file.

Dynamic content contains variables that will be randomly selected by the Randori engine to be include in the output file.

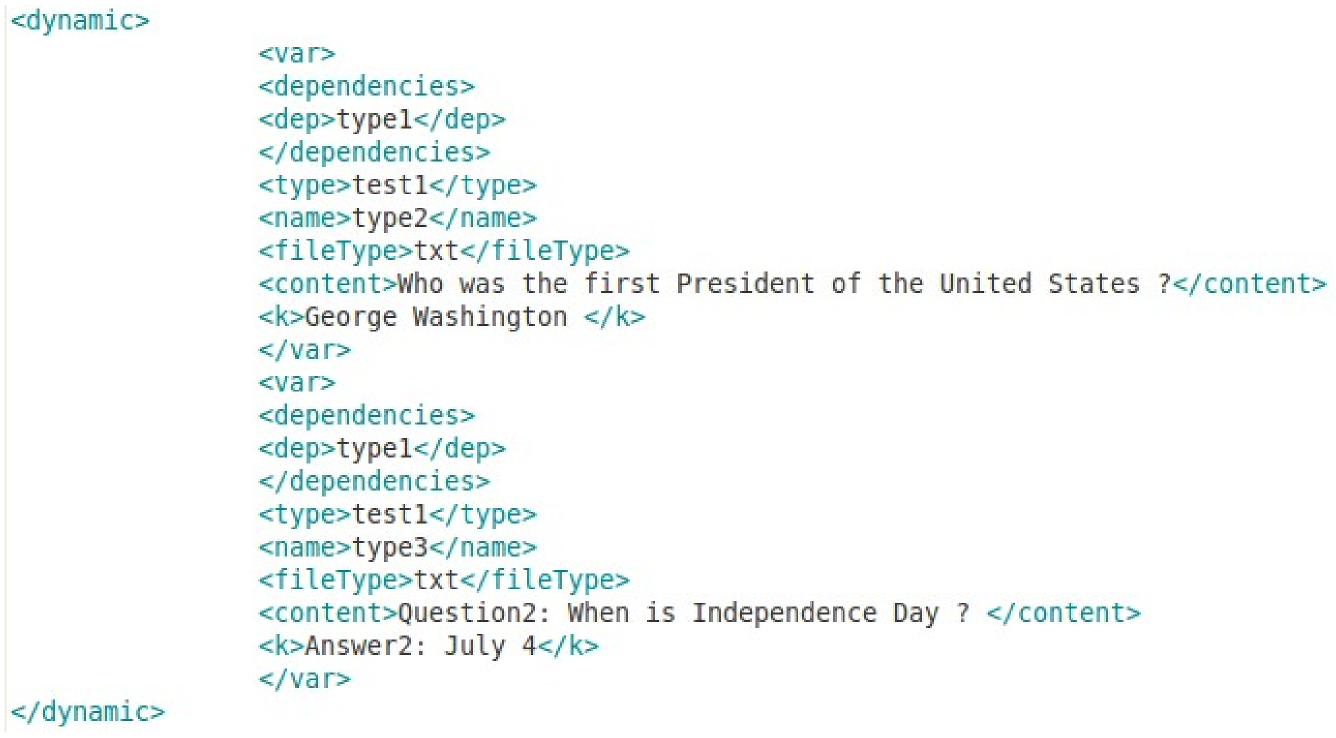
Each possible variable has the following attributes:

- type - type of content
- name - the name of the specific piece of content
- file type -file the content should be included
- dependencies – the names of other content that must be present in the content for a piece of content to be included
- content – the data to be include in the output file
- key – the data related to the data to be included in the output to be include in the key file

These dynamic variables are loaded into an array. One of them is then randomly chosen from that array and loaded into an array of content to be included in the output.

Dynamic content example 1:

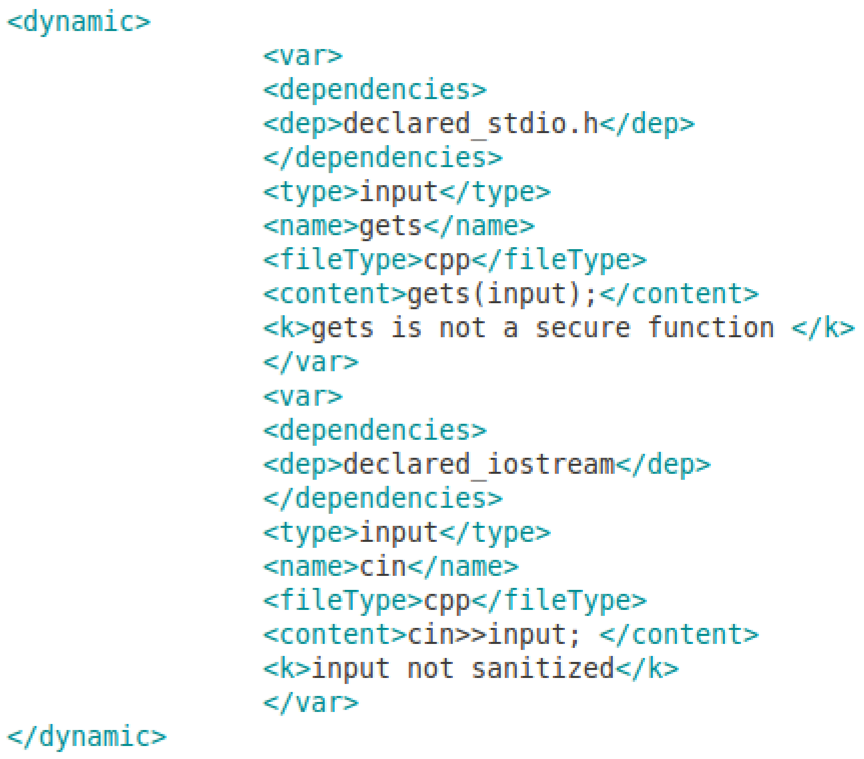
In this example there are two possible variables. These variables are for a set of questions. These variables both have one dependency named type1. These variables will only be written into the output file if the variable type1 has already been chosen by the Randori engine to be in the output. They are type test1 and will appear anywhere in the output that the test1 variable is defined as occurring in the template. The variables are named type2 and type3. They are questions in the content tag with their answers in the k tag.



```
<dynamic>
                <var>
                <dependencies>
                <dep>type1</dep>
                </dependencies>
                <type>test1</type>
                <name>type2</name>
                <fileType>txt</fileType>
                <content>Who was the first President of the United States ?</content>
                <k>George Washington </k>
                </var>
                <var>
                <dependencies>
                <dep>type1</dep>
                </dependencies>
                <type>test1</type>
                <name>type3</name>
                <fileType>txt</fileType>
                <content>Question2: When is Independence Day ? </content>
                <k>Answer2: July 4</k>
                </var>
</dynamic>
```

Dynamic content example 2:

In this example there are two possible variables. These variables are for creating an application written in the C++ programming language. The first variable depends on the variable declare_stdio.h being present. The second variable depends on the variable declared_iostream being declared. These variables will only be written into the output file if the correct variables have already been chosen by the Randori engine to be in the output. The variables are named for the functions that will be inserted in to the output. They are lines of source code to be inserted into the content tag with their security issues in the k tag.



```
<dynamic>
                <var>
                <dependencies>
                <dep>declared_stdio.h</dep>
                </dependencies>
                <type>input</type>
                <name>gets</name>
                <fileType>cpp</fileType>
                <content>gets(input);</content>
                <k>gets is not a secure function </k>
                </var>
                <var>
                <dependencies>
                <dep>declared_iostream</dep>
                </dependencies>
                <type>input</type>
                <name>cin</name>
                <fileType>cpp</fileType>
                <content>cin>>input; </content>
                <k>input not sanitized</k>
                </var>
</dynamic>
```

Dynamic content example 3:

The Randori engine by default creates a variable called “fileNum” with the name of the newly created file. This will be needed if a file needs to make a reference to itself. For example, a java file needs to contain a class with the same name as the name of the file. The process for adding these references is included in the addVar section below.

Static

The static tag is used to define static content. Static content is written to each file created by the Randori engine.

Static content has the following attributes:

- content - the data to be include in the output file
- file type -file the content should be included

Example 1:

```
<static>
<fileType>txt</fileType>
<content>
Please answer all of the following questions:
</content>
</static>
```

This static content contains instructions that will be included in a txt file.

Example 2:

This static content is source code that will be written by the Randori engine to to a .py file and compilation and execution instructions that will be written to a README file.

```
<static>
<fileType>py</fileType>
<content>print "hello world"</content>
</static>
```

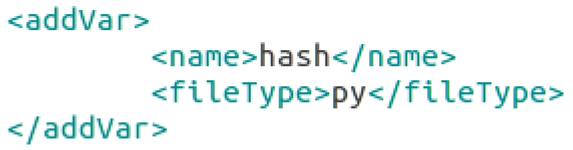
addVar

The addVar tag is used to define where dynamic variables should be added and what file those variables should be added to. The information from the dynamic variables with the name specified by the addVar tag will be written to the files created by the Randori engine .

addVar content has the following attributes:

- name - the data to be include in the output file
- file type -file where the content should be included

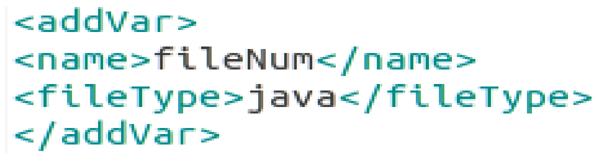
Example 1:
Here a Randori template is specifying the adding of a variable that specifies a hashing algorithm named “hash” to be used by the application. The Randori engine will add the variable that was chosen when the dynamic variables were created for this specific file.



```
<addVar>
        <name>hash</name>
        <fileType>py</fileType>
</addVar>
```

Example 2:

Here a Randori template is specifying the adding of the file's name. The Randori engine by default creates a variable called “fileNum” with the name of the newly created file. This will need to be done if the name of the file needs to be the same as the name of the class.



```
<addVar>
<name>fileNum</name>
<fileType>java</fileType>
</addVar>
```

Diagnostic engine

For users who want to create their own Randori Diagnostic Engine templates we have included the Randori Diagnostic Engine. The diagnostic engine will display what variables are being created and added to the files created b the engine. This application will create one file from a user's XML template. This template will need to be changed in the source code of the application .

The application can be run using the following command:

```
user$ python randori.py –i authN.xml –d
```

Templates Included with This Release

samplePasswordPolicy.xml – This template generates vulnerable password policies for a randomly named organization in the health care or defense industry as a text file.

authN.xml – This template generates a vulnerable application that authenticates users written in the python programming language. The application requests a user's user-name and password and returns whether or not they have been authenticated.

Proof of concept templates are included with this release for the java and python programming languages. Proof of concept code is also included for creating quizzes.

Project Goal and Vision

The goal of this project is to create a training tool that provides those wanting to learn applied information security skills with a no-cost, safe, practical, and effective way to get hands on experience. The vision for this project is that the open-source, modular nature of the project will enable educators, professionals, and developers to add their own content.

About

Randori is a martial arts term that means “chaos taking” and refers to free-style practice. The randori concept has been adapted into this training tool. Like the randori of marital arts, this program provides students with randomized files to learn from to help better prepare them for real applications or policies they will encounter as security professionals.