Implement a Defensive Security System

This assignment will help you understand security mechanisms. You will be guided through the steps of creating a reference monitor using the security layer functionality in Repy V2. A reference monitor is an access control concept that refers to an abstract machine that mediates all access to objects by subjects. This can be used to allow, deny, or change the behavior of any set of calls. While not a perfect way of validating your reference monitor, it is useful to create test cases to see whether your security layer will work as expected. (The test cases may be turned in as part of the next assignment.)

This assignment is intended to reinforce concepts about access control and reference monitors in a hands-on manner.

Table of Contents

Overview

In this assignment you will create a security layer which stops the user from writing to a file after a specified file size. You will add a function called setmaxfilesize() that will take a file size as parameter

and enforce it for the specified file. If a file is longer than the specified size, any data past this point in the file will be truncated. (Since the Repy V2 API does not have a truncate call, you will have to implement this functionality yourself.) If a writeat() call is performed at a location at or past the end of the file size limit, a SeekPastEndOfFileError should be raised. If the writeat() starts before the file limit, any data up to the size limit should be written and all other data should be discarded.

Three design paradigms are at work in this assignment: accuracy, efficiency, and security.

- Accuracy: The security layer should only stop certain actions from being blocked. All other actions should be allowed. For example, If a user tries to read data or write data with in file size limit, then it should not stop user from doing it.
- Efficiency: The security layer should use a minimum number of resources, so performance is not compromised.
- Security: The attacker should not be able to circumvent the security layer. Hence, if the data can be written beyond specified size, the security is compromised.

Getting Python

Please note you must have Python 2.5, 2.6, or 2.7 to complete this assignment. Instructions on how to get Python for Windows can be found here. If you are using Linux or a Mac it is likely you already have Python. In order to verify this, simply open a terminal and type `python`. Please check its version on the initial prompt.

Note: If you are using Windows, you will need python in your path variables. A tutorial on adding variables to your path in Windows can be found at http://www.computerhope.com/issues/ch000549.htm

Getting RepyV2

The preferred way to get Repy is to download the sandbox as a zipfile. Unzip this and copy your files to unzipped directory. Use the command found below in order to run Repy files:

```
python repy.py restrictions.default encasementlib.r2py [security_layer].r2py [program].r2py
```

Please note Repy files end in extension `.r2py`.

In order to test whether or not these steps worked, please copy and paste the code found below for the sample security layer and sample attack. Where you can find the sample security layer and sample attack:

- Sample security layer: BasicAndInadequateDefense
- Sample attack layer: Testingyoursecuritylayer

If you got an error, please go through the trouble shooting section found below.

Troubleshooting Repy code

If you can't get Repy files to run, some of the following common errors may have occurred:

- using `print` instead of `log`:

Repy is a subset of Python, but its syntax is slightly different. For example, Python's `print` statement cannot be used; Repy has `log` for that. For a full list of acceptable syntax please see wiki:RepyV2API.

- command line errors:

files are missing: In the above command line call, you must have `repy.py`, restrictions.default, encasementlib.r2py, the security layer and the program you want to run in the current working directory. If any or all of the above files are not in that directory then you will not be able to run repy files.

Tutorials for Repy and Python

Now that you have Repy and Python, you may need a refresher on how to use them. The following tutorials are excellent sources of information.

- Python tutorial: **http://docs.python.org/tutorial/**

- Seattle tutorial: **https://seattle.poly.edu/wiki/PythonVsRepy**
- list of RepyV2 API calls: **RepyV2API**

Building the security layer

The following program is a basic and incomplete sample code for you to get an idea about writing security layer. Remember, you have no idea how the attacker will try to penetrate your security layer, so it is important that you leave nothing to chance!

A basic (and inadequate) defense

Time to start coding! Let's inspect a basic security layer.

```
"""
This security layer interposes size restrictions on a file.  If a call is made
to setmaxfilesize, the file must not be allowed to grow larger than the
specified value.  If a user tries to write that starts before the end of file
and extends past the allowed size, bytes past the allowed size are discarded.
If a write starts at or past the allowed size, a SeekPastEndOfFileError must
be raised.  If a file already contains more than the maximum file size bytes
when the call is made, the file must be truncated so that it is of the
appropriate size. It should also raise ValueError if the size to be set is not
valid(e.g. less than 0 or non-number )



Note:
    This security layer uses encasementlib.r2py, restrictions.default, repy.py and Pytho
    Also you need to give it an application to run.
    python repy.py restrictions.default encasementlib.r2py [security_layer].r2py [attack

    """


class SecureFile():
```

```
  def __init__(self,file):
    # globals
    mycontext['debug'] = False
    # starts as "None", which allows writes of any size.
    mycontext['size'] = None
    # local (per object) reference to the underlying file
    self.file = file

  def setmaxfilesize(self,size):
    mycontext['size'] = size

  def writeat(self,data,offset):

    #raise error if user try to write beyond permitted file size
    if mycontext['size'] and offset >= mycontext['size']:
      raise SeekPastEndOfFileError("Tried to write past the end!")

    # Write the requested data to the file using the sandbox's writeat call
    self.file.writeat(data,offset)

  def readat(self,bytes,offset):
    return self.file.readat(bytes,offset)

  def close(self):
    return self.file.close()


def secure_openfile(filename, create):
  f = openfile(filename,create)
  return SecureFile(f)

def secure_listfiles():
  return listfiles()

def secure_removefile(filename):
  return removefile(filename)



# The code here sets up type checking and variable hiding for you.  You should not need
```

```
sec_file_def = {"obj-type":SecureFile,
                "name":"SecureFile",
                "setmaxfilesize":{"type":"func","args":(int,long, type(None)),"exception
                "writeat":{"type":"func","args":(str,(int,long)),"exceptions":Exception,
                "readat":{"type":"func","args":((int,long,type(None)),(int,long)),"excep
                "close":{"type":"func","args":None,"exceptions":None,"return":(bool,type
           }

CHILD_CONTEXT_DEF["openfile"]["target"] = secure_openfile
CHILD_CONTEXT_DEF["listfiles"]["target"] = secure_listfiles
CHILD_CONTEXT_DEF["removefile"]["target"] = secure_removefile

# Execute the user code
secure_dispatch_module()
```

Testing your security layer

In this part of the assignment you will pretend to be an attacker. Remember the attacker's objective is to bypass the file size restrictions or cause the security layer to act in a disallowed manner. By understanding how the attacker thinks, you will be able to write better security layers.

An example of an attack is found below:

```
if "testfile.txt" in listfiles():
  removefile("testfile.txt")
myfile=openfile("testfile.txt",True)  #Open a file

# put some initial data in the file.
myfile.writeat("test123456",0)

#set maximum file size allowed to 10
myfile.setmaxfilesize(10)

try:
  myfile.writeat("test123456abcdefg",1000)
except SeekPastEndOfFileError:
```

```
  pass # there should be an exception here...
else:
  log("ERROR!  Didn't stop me from writing past the maximum file size.\n")


#Close the file
myfile.close()
```

Note: All attacks should be written as Repy V2 files, using the .r2py extension.

Choice of File Names

It is important to keep in mind that only lowercase file names are allowed. So in the above code, specifically:

```
# Open a file
myfile=openfile("look.txt",True)
```

look.txt is a valid file name, however Look.txt is not. Examples of other invalid files names are, look@.txt, look/.txt, and look().txt. Essentially all non-alphanumeric characters are not allowed.

Running your security layer

Finally, type the following commands at the terminal to run your security layer with your attack program

```
python repy.py restrictions.default encasementlib.r2py [security_layer].r2py
[attack_program].r2py
```

Make sure you went through the "How to get RepyV2" section!

Notes and Resources

- For a complete list of syntax in Repyv2 please visit: **RepyV2API**
- The following link is an excellent source for information about security layers: **http://isis.poly.edu/~jcappos/papers/cappos_seattle_ccs_10.pdf**
- **Note:** It is possible to add multiple security layers to Repy, this may be useful for testing different mitigations separately. This is done with the following command at the terminal:

```
python repy.py restrictions.default encasementlib.r2py [security_layer1].r2py
[security_layer2].r2py [security_layer3].r2py [program].r2py
```

Your security layer should produce no output!!

- In repy log replaces print from python. This may be helpful when testing if Repy installed correctly.

Extra Credit

For extra credit, make a second security layer which retains its specified size limits even if it is closed and reopened again. The file size limit should stay after the program is terminated and restarted Do not submit this code inside your assignment. Submit a separate copy for extra credit.

What to turn in?

- Turn in a repy file called reference_monitor_[netid].r2py with all letters in lowercase.

* **Never raise unexpected errors or produce any output.** Your program must produce no output when run normally.

- For extra credit turn in a second repy file called extra_credit_[netid].r2py **You must turn in separate files for the normal assignment and extra credit**