SIL765: Assignment 2 on Heap Spraying Attacks

Goal

In the first project, you gained experience with exploiting memory vulnerabilities in a care- fully controlled sandbox. In this project, you will experience the added difficulty of exploiting a large program-Firefox-in a less carefully controlled setting.

Your goal will be to produce an HTML file that, when opened in a vulnerable Firefox browser, executes arbitrary machine code with the privileges of the user running Firefox. Specifically, we will ask you to run code that binds a shell to port 8080 on the victim's machine.

As with the previous project, your work must be done on the VMware virtual machine provided on the course website; see below for information about this environment.

To save you the minutes and minutes of effort that would be required to find an exploitable vulnerability in Firefox, we have packaged a custom version of Firefox with the Boxes VM you'll use for this project. This special version has a new DOM method, navigator.pwnMe(), that will transfer control to the memory address given to it as an argument.

Just calling this method won't be enough. You'll be graded on the reliability of your exploit. Your exploit should work when the browser is freshly opened, and after a long browsing session. Your exploit should work with one tab open, or dozens. To make things worse, Firefox will be running with address-space layout randomization (ASLR) enabled, so objects may move in memory between otherwise identical runs.

To make your exploit reliable under such challenging conditions, you will want to use heap spraying.

Note: In this project, you are being asked to perform exploitation of real-world programs under real-world conditions, not the more controlled laboratory conditions of the first project. You may find this project less straightforward and more open-ended than the first one.

The Vulnerable Browser

For reasons of licensing and philosophy that are mostly orthogonal to our concerns, the Debian Project's version of the Firefox browser does not use the Firefox name or branding. Instead, the browser is called "Iceweasel".

Running iceweasel on the command line from an xterm will launch a browser window. Debug symbols are available in case you find them helpful. If you run iceweasel -g on the command line, Iceweasel will run itself under GDB.

The Firefox browser keeps its configuration, history, and caches in the ~/.mozilla/directory. You can remove this directory (or rename it) when you wish to start fresh.

For your reference, here is the C++ code implementing the navigator.pwnMe() DOM method:

```
NS_IMETHODIMP

nsNavigator::PwnMe(PRUint32 addr)

{

((void (*)())(addr))();

return NS_OK;

}
```

The Environment

You (and we, for grading!) will test your exploit programs within a VMware virtual machine. The Boxes environment for this project has the X Win- dowing System installed. (You'll need X to run Firefox.) You can use SSH X forwarding to have X clients running insider the VM display on the host machine.

The Assignment

You are to write a single HTML file, called [yourlastname].html. This file must be self-contained: it must not include any external scripts or other resources.

When opened in the Iceweasel browser (using File -> Open from the menu), this file should exploit the pwnMe vulnerability and bind a shell to port 8080 on the VM. We will then connect to this port from the host machine and run shell commands, using netcat: "nc [VM IP address] 8080". To get you started, we have provided a bindshell shellcode that execs netcat to bind a shell to port 8080, as required, at bindshell.txt; feel free to modify this shellcode or use a

different one if you prefer.

We will test your exploit in several configurations, and grade you based on its reliability. You can use your HTML file to display a witty message to the TAs, if you'd like. Exploits that are stealthy that do not visibly interfere with the user's browsing session can earn a small amount of extra credit.

Hints

1. You can find the process ID for running Iceweasel processes by running "pstree -p". The child processes shown will in fact be Linux "light-weight processes" (threads), and won't have their own memory map. For the parent process, you can show its memory map by listing the file "/proc /[processid]/maps".
2. There doesn't seem to be any convenient way to examine the internals of the JavaScript heap from within Firefox itself. If you do want to use GDB to explore the JavaScript heap, you will find it handy to have access to a JavaScript context object, cx. Here are instructions, for obtaining a context:
   a. Run Firefox in debug mode, using the command "iceweasel -g".
   b. Create a breakpoint at a function that is very rarely called on Firefox:

      (gdb) break js_ValueToXMLObject

      If asked whether to make this breakpoint pending on library load, say yes. This is a function for XML processing. Inside this function, we can get the access to cx, the JavaScript context.

   c. Do your heap spray. At the JS program point that you believe your heap spray is done, insert a JavaScript statement such as the following (to trigger a call to js_ValueToXMLObject):

      var sales = (<) sales vendor="John">

      (<)item type="peas" price="4" quantity="6"/>

      (<)item type="carrot" price="3" quantity="10"/>

      (<)item type="chips" price="5" quantity="3"/>

      (<)/sales>;

      The JS code triggers the function with the breakpoint.

   d. In the GDB breakpoint inside the function, the variable cx is a usable JavaScript context.

3. Test your exploit in many different settings: with Iceweasel freshly launched and after some browsing, with few or many tabs open, with ~/.mozilla/ present and without, etc.

Files

- Virtual machine image: boxes2x-vuln.tar.bz2
- Bindshell: bindshell.txt
  - Untar it.
  - login="root", password="root"
  - login="user", password="user"

Deliverables

1. You are to provide a tarball (i.e., a .tar.gz or .tar.bz2 file) containing the HTML file for your exploit.
2. There should be no directory structure: all files in the tarball should be in its root directory.
3. Along with your HTML exploit file, you must include a text file to explain the strategy you used in your exploit.
4. Finally, you must include file called ID which contains, on a single line, the following: your Rollno and your name, in the format last name, comma, first name.

READINGS: Heap Spraying / Stack Spraying Attacks [PPT]

1. Nozzle: A Defense Against Heap-Spraying Code Injection Attacks, P. Ratanaworabhan et al. (MSR). November 2008

Note:

1. To be done individually.
2. The last date of submission of is **14th Feb, 11:59:00 PM**.
