ZIP 作成時の UNICODE によって分離された文字を使ったサニタイズ回避法について

ＮＴＴ コミュニケーションズ株式会社
ソリューションサービス部
第四エンジニアリング部門
セキュリティオペレーション担当

2011 年 10 月 19 日

Ver. 1.3

1. 調査概要

Web アプリケーションなどで ZIP ファイルを生成する際、圧縮対象のファイル名を適切にサニタイズしないと、不用意に「..\」などの文字列が混入し、脆弱性を有する古い展開ツールを使っている利用者がそのファイルを展開する際に、予期せぬディレクトリ上にファイルが展開される可能性があることを検証した。
システム開発時に圧縮ライブラリを使用する際、ファイル名の文字コードについても注意した上で、システム開発を行う必要がある。

2. Webアプリケーションと ZIP作成機能

2.1. Webアプリケーションと ZIP作成機能

Web アプリケーションによって、ファイルをアップロードする機能を有する場合、稀にアップロードしたファイルを ZIP 圧縮した状態で、ダウンロード可能となる Web サイトがある。
ZIP 圧縮することで、通信量の削減や、Microsoft 社の Web ブラウザ Internet Explorer のファイル内容で判断する機能による XSS 誘発を防ぐなどを目的としていると思われる。

また、Windows上ではZIP形式で圧縮されたファイルのファイル名には、UNICODEではなくANSIコードを使用しているため、UNICODEによって分離された文字(円記号[u00A5])を使ったサニタイズ回避テクニックを用いられることで、ZIP圧縮する際のサニタイズ処理が回避され不正なファイル名が忍び込む危険性がある(図 2.1-2～図 2.1-4)。

一部の展開ツールでは、セキュリティ侵害行為とならないように、予期しない場所へのファイルの展開はできないようになっている(図 2.1-5～図 2.1-8)。

逆に一部の展開ツールでは、圧縮されたファイル名の指示通りに展開してしまうツールも未だに存在する(図 2.1-9～図 2.1-10)。

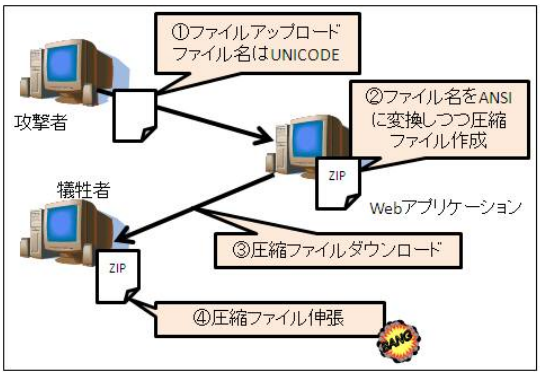

図 2.1-1 : 攻撃シナリオは、UNICODE のファイル名でアップロードした圧縮ファイルが、犠牲者のホスト上で伸張される際に、ディレクトリ・トラバーサル攻撃が成立する可能性がある

図 2.1-2 :「文字コード表」を使って、UNICODE の円記号をファイル名に使う

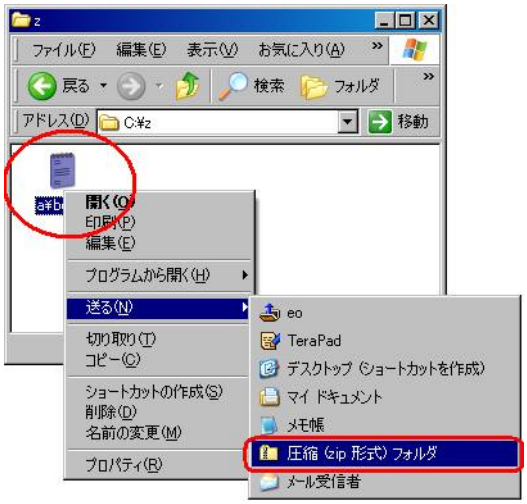

図 2.1-3 : ファイル名に UNICODE で分離された円記号を含むファイルを
MS-WindowsXP SP3 標準の「ZIP フォルダ」で圧縮してみる

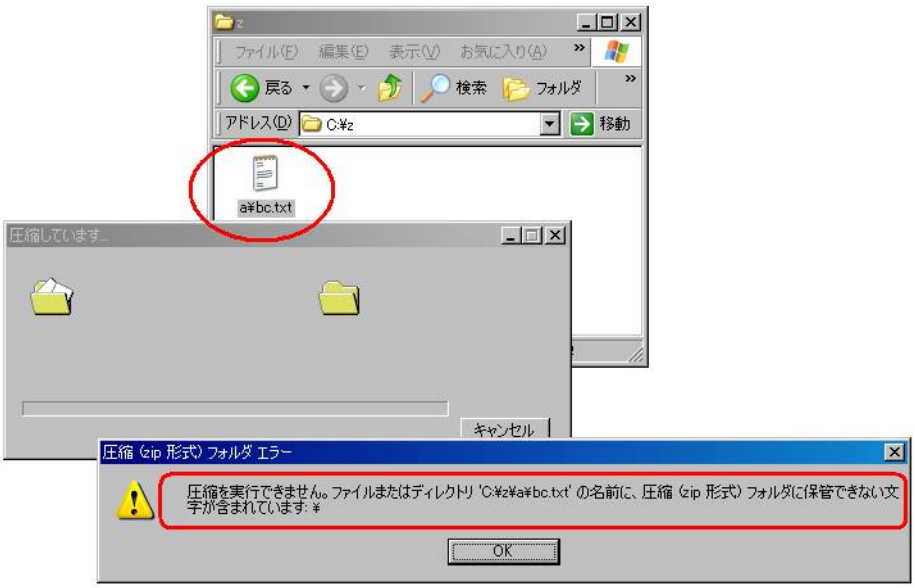

図 2.1-4 : 図 2.1-3の結果。UNICODEによって分離された円記号を
ファイル名に含むファイルは圧縮できないようだ

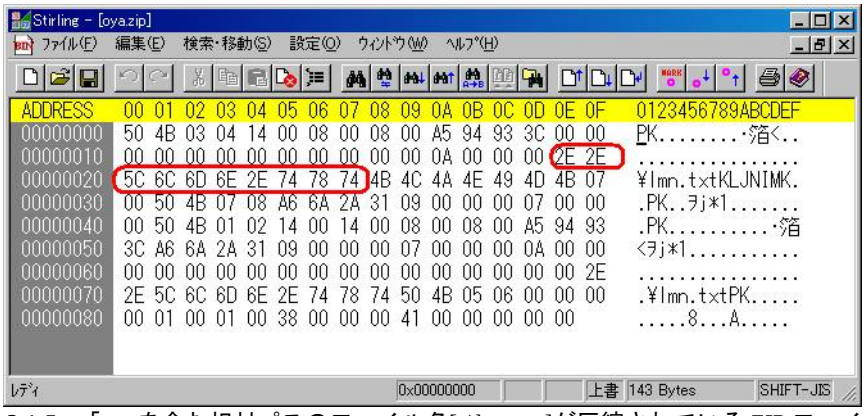

図 2.1-5 : 「..」を含む相対パスのファイル名[..\lmn.txt]が圧縮されている ZIP ファイル

図 2.1-6 : 図 2.1-5をデスクトップ上に展開しようとするとeo1.5.2 は
「..」を無視し、そのままデスクトップ上に展開した

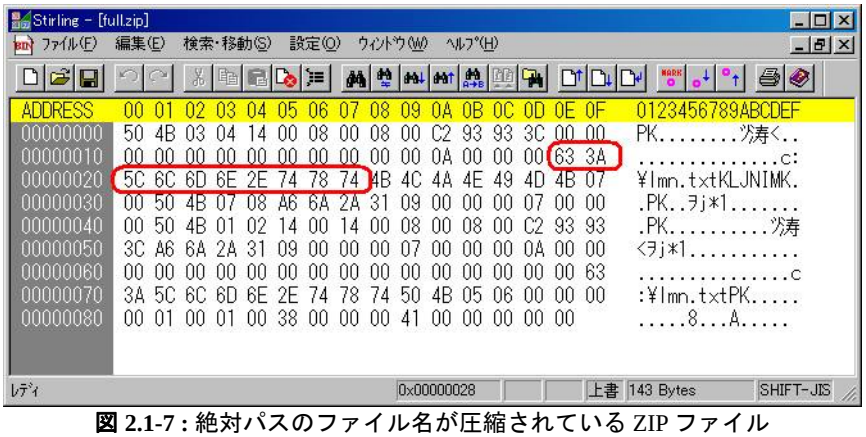

図 2.1-7 : 絶対パスのファイル名が圧縮されている ZIP ファイル

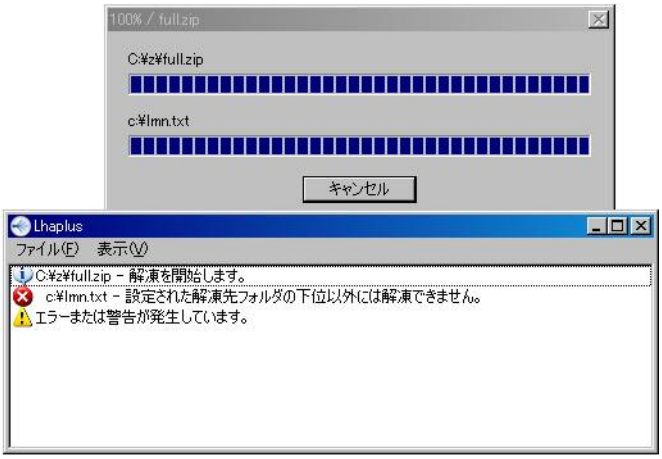

図 2.1-8 : 図 2.1-7をデスクトップ上に展開しようとするとLhaplus1.57 は
このようなエラーが表示されて、展開に失敗する

図 2.1-9 : 相対パス形式となっている図 2.1-5を今度は、Lhaplus1.57 で、
ZIPファイルのあるディレクトリ上(c:\x\y)に展開してみる

図 2.1-10 : 図 2.1-9の結果。Lhaplusは相対パス形式については、
指示通り([c:\x\y]の一つ上のディレクトリ[..\lmn.txt]なので、[c:\x])に展開するようだ

3. Javaの場合

Java の場合、JDK 標準の java.util.zip を使うことで、ZIP アーカイバを扱うことができる。
しかしながら、圧縮ファイル名は UTF-8 に変換されてしまうため、Windows 上で展開する場合、日本語文字を含むファイル名が文字化けしてしまう。
Java には、JDK 標準以外にも、Ant パッケージの org.apache.tools.zip を使うことでも ZIP アーカイバを扱うことができる。こちらは、文字コードを指定できるなど、JDK 標準より柔軟である。
この Ant を使用して ANSI 文字コードのファイル名として圧縮処理を行う場合、UNICODE によって分離された文字を使ったサニタイズ回避テクニックが有効なため、ファイル名を一度 ANSI コードへ変換し、その上で UNICODE に変換しなおした上(Java コード上の文字コードは UNICODE であるため)でサニタイズ処理を行うことが求められる。

3.1. java.util.zipの場合

検証環境

- MS-WindowsXP SP3
- JDK 1.6.0_06

```
import java.io.File;
import java.io.FileOutputStream;
import java.util.zip.ZipOutputStream;
import java.util.zip.ZipEntry;
// import org.apache.tools.zip.ZipOutputStream;
// import org.apache.tools.zip.ZipEntry;

public class ZipTest{
 public static void main(String[] argv){
  int i;
  System.out.println("usage : ZipTest ZIPFileName FileNameString ArchivedData Flg");
  System.out.println(" Flg = 1 -> Replace[u005c]->[u00a5]");
  if(2 < argv.length){
   String myData = argv[2];
   String myFileNameStr = argv[1];
   byte[] myDataHako = myData.getBytes();

   if(3 < argv.length){
    System.out.println("Before String = " + myFileNameStr);
    char[] tempHako = myFileNameStr.toCharArray();
    for(i=0;i<tempHako.length;i++){
     System.out.print(tempHako[i] + "(" + Integer.toHexString((int)tempHako[i]) + ") ");
     if(Character.valueOf(tempHako[i]).equals('¥¥') == true){
      char[] t = Character.toChars(165);
      tempHako[i] = t[0];
     }
    }
    myFileNameStr = "";
    for(i=0;i<tempHako.length;i++){
     myFileNameStr += Character.toString(tempHako[i]);
    }
    System.out.println("¥nAfter String = " + myFileNameStr);
    tempHako = myFileNameStr.toCharArray();
    for(i=0;i<tempHako.length;i++){
     System.out.print(tempHako[i] + "(" + Integer.toHexString((int)tempHako[i]) + ") ");
    }
    System.out.println("");
   }
   try{
    File zipFile = new File(argv[0]);
    ZipOutputStream myZipOutputStream = new ZipOutputStream(new FileOutputStream(zipFile));
//    myZipOutputStream.setEncoding("MS932");
    myZipOutputStream.putNextEntry(new ZipEntry(myFileNameStr));
    myZipOutputStream.write(myDataHako);
    myZipOutputStream.closeEntry();
    myZipOutputStream.close();
   }catch(Exception e){
    e.printStackTrace();
   }
  }
  System.out.println("End");
 }  }
```

図 3.1-1：検証コード(java.util.zip[JDK 標準]の場合)

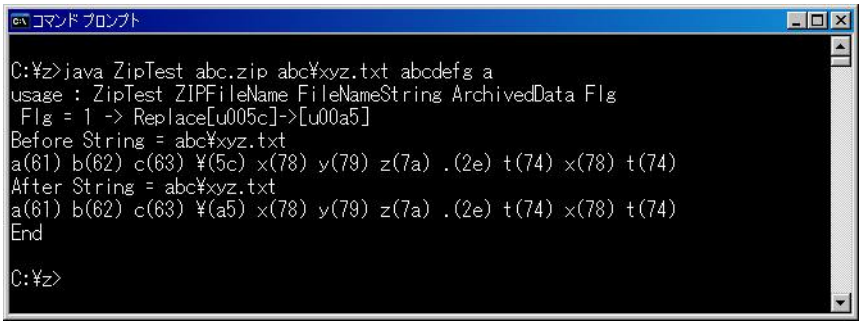

図 3.1-2 : 図 3.1-1の実行結果

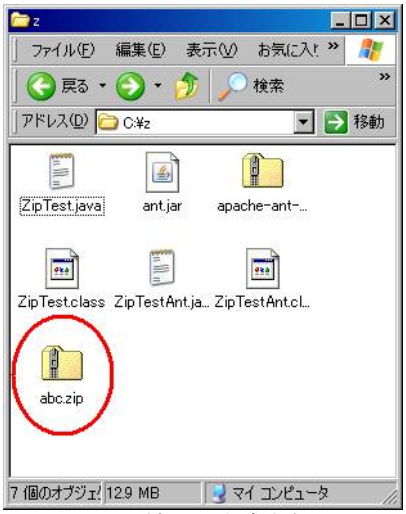

図 3.1-3 : 図 3.1-2の結果、作成されたZIPファイル

図 3.1-4 : 図 3.1-3の中身をMS-WindowsXP標準の「ZIPフォルダ」で中身を確認すると
ファイル名文字化けしている

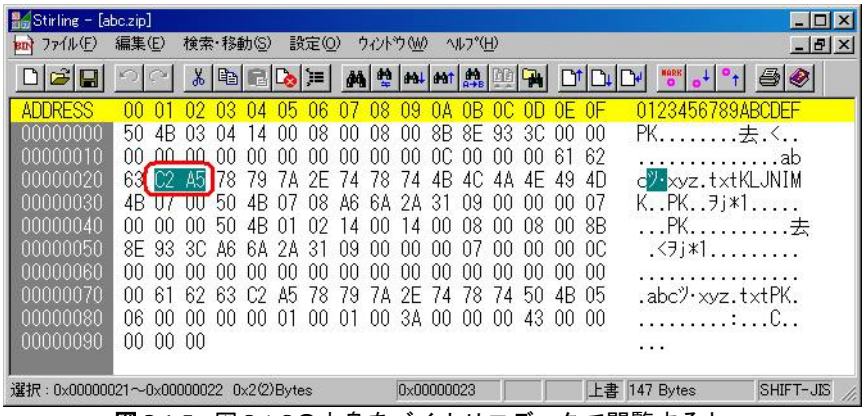

図 3.1-5 : 図 3.1-3の中身をバイナリエディタで閲覧すると、

「円記号」が「c2 a5」(UTF-8)となっている

3.2. org.apache.tools.zip(Antパッケージ)の場合

検証環境

- MS-WindowsXP SP3
- JDK 1.6.0_06
- Ant 1.8.0

```
import java.io.File;
import java.io.FileOutputStream;
// import java.util.zip.ZipOutputStream;
// import java.util.zip.ZipEntry;
import org.apache.tools.zip.ZipOutputStream;
import org.apache.tools.zip.ZipEntry;

public class ZipTest{
 public static void main(String[] argv){
  int i;
  System.out.println("usage : ZipTest ZIPFileName FileNameString ArchivedData Flg");
  System.out.println(" Flg = 1 -> Replace[u005c]->[u00a5]");
  if(2 < argv.length){
   String myData = argv[2];
   String myFileNameStr = argv[1];
   byte[] myDataHako = myData.getBytes();

   if(3 < argv.length){
    System.out.println("Before String = " + myFileNameStr);
    char[] tempHako = myFileNameStr.toCharArray();
    for(i=0;i<tempHako.length;i++){
     System.out.print(tempHako[i] + "(" + Integer.toHexString((int)tempHako[i]) + ") ");
     if(Character.valueOf(tempHako[i]).equals('¥¥') == true){
      char[] t = Character.toChars(165);
      tempHako[i] = t[0];
     }
    }
    myFileNameStr = "";
    for(i=0;i<tempHako.length;i++){
     myFileNameStr += Character.toString(tempHako[i]);
    }
    System.out.println("¥nAfter String = " + myFileNameStr);
    tempHako = myFileNameStr.toCharArray();
    for(i=0;i<tempHako.length;i++){
     System.out.print(tempHako[i] + "(" + Integer.toHexString((int)tempHako[i]) + ") ");
    }
    System.out.println("");
   }
   try{
    File zipFile = new File(argv[0]);
    ZipOutputStream myZipOutputStream = new ZipOutputStream(new FileOutputStream(zipFile));
    myZipOutputStream.setEncoding("MS932");
    myZipOutputStream.putNextEntry(new ZipEntry(myFileNameStr));
    myZipOutputStream.write(myDataHako);
    myZipOutputStream.closeEntry();
    myZipOutputStream.close();
   }catch(Exception e){
    e.printStackTrace();
   }
  }
  System.out.println("End");
 }  }
```

図 3.2-1：検証コード(org.apache.tools.zip[Ant]の場合)

文字コードを指定するメソッド以外、図 3.1-1とほとんど同じである

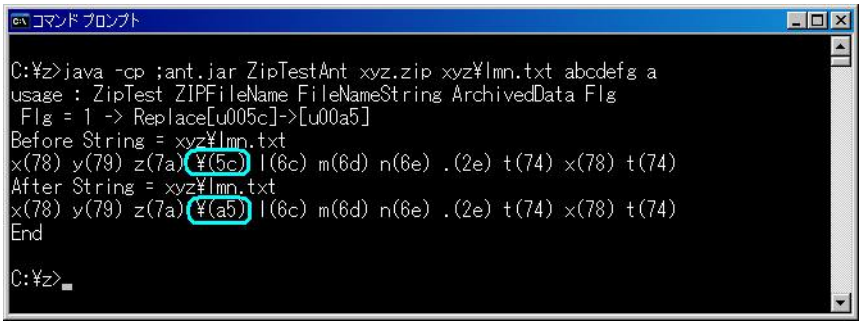

図 3.2-2 : 図 3.2-1の実行結果

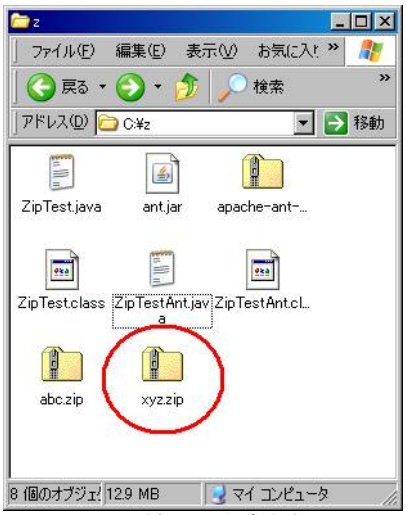

図 3.2-3 : 図 3.2-2の結果、作成されたZIPファイル

図 3.2-4 : 図 3.2-3の中身をMS-WindowsXP標準の「ZIPフォルダ」で中身を確認すると
サブフォルダが存在している

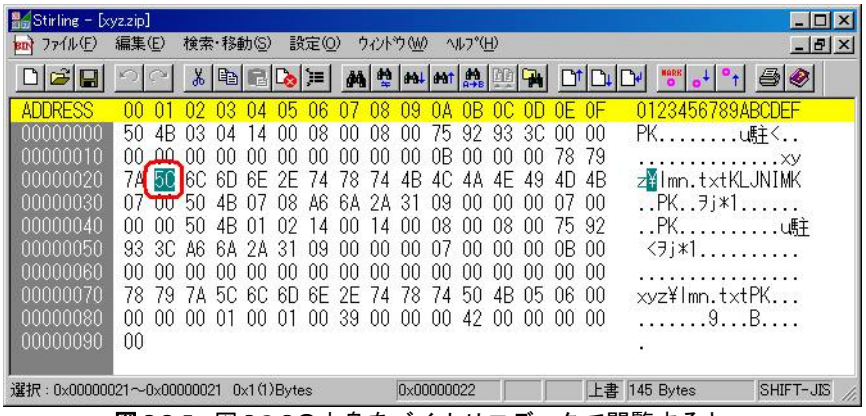

図 3.2-5 : 図 3.2-3の中身をバイナリエディタで閲覧すると、
「円記号」が「5c」(つまり 0x5cに縮退している)となっている

4. PHPの場合

PHP で、ZIP 圧縮を行う場合、PHP 標準の zip 圧縮ライブラリ(Win32 版：php_zip.dll)と、phpMyAdmin がインストールされた環境であれば、phpMyAdmin のライブラリ(zip.lib.php)を使う場合と、二通りの方法がある。

これらを使って、ファイル・アップロード機能があり、アップロードされたファイルを ZIP 圧縮する UTF-8(UNICODE)の Web ページを作成し、挙動の確認を行った。

検証の結果、標準の zip 圧縮ライブラリ、phpMyAdmin のライブラリ共に、ファイル名の文字コードを自動変換する機能を有していないことを確認した。
よって、共に ZIP ファイル内部の圧縮されたファイルのファイル名は、与えられた文字コード UTF-8 のままであり、本文書が期待するサニタイズ回避テクニックは使用することができないことを確認した。
つまり、プログラマが明示的に文字コード変換処理を行う必要があり、その際に文字コード変換前にサニタイズ処理を行うようなコーディングをしていれば、本文書が期待するサニタイズ回避テクニックが有効に働くものと思われるが、プログラマのそのような行動は、稀であると思われる。

4.1. php_zip.dll (Win32) の場合

検証環境

- MS-Windows2000 SP4
- Apache 2.2.11 for Win32
- PHP 5.2.13 for Win32

```
<html>
         <head>
          <meta http-equiv="content-type" content="text/html;charset=utf-8">
          <title>test1</title>
         </head>
         <body>
<?php
$display_data = "";
$zip_name = "test1.zip";
if(isset($zip_name)){

        if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
                $filename = $_FILES['upfile']['name'];
                $file_content = $_FILES['upfile']['tmp_name'];
                $contents = file_get_contents($file_content);
                $zip = new ZipArchive();
                $result = $zip->open($zip_name , ZipArchive::CREATE);

                if($result === TRUE){
                        $zip->AddFromString($filename , $contents);
                        $zip->close();
                }
                $hex_content = bin2hex($filename);
                $count = strlen($hex_content) / 2;

                for($i=0;$i<$count;$i++){
                        if($i==0){
                                $t=0;
                        }else{
                                $t=$i * 2;
                        }
                        $data = substr($hex_content , $t , 2);
                        $display_data = $display_data . $data . " ";
                }
        }
}
?>
                <form enctype="multipart/form-data" method="post" action="">
                アップロード：<br>
                <input type="file" name="upfile"><br>
                <input type="submit">
                </form>
                <hr>
                <?php echo $filename; ?><br>
                <?php echo $display_data; ?>
        </body>
</html>
```

図 4.1-1：検証コード

図 4.1-2 : ZIP ファイルの保存先フォルダ

図 4.1-3 : 図 4.1-1のWebページ、ここでtest.txtをアップロードする

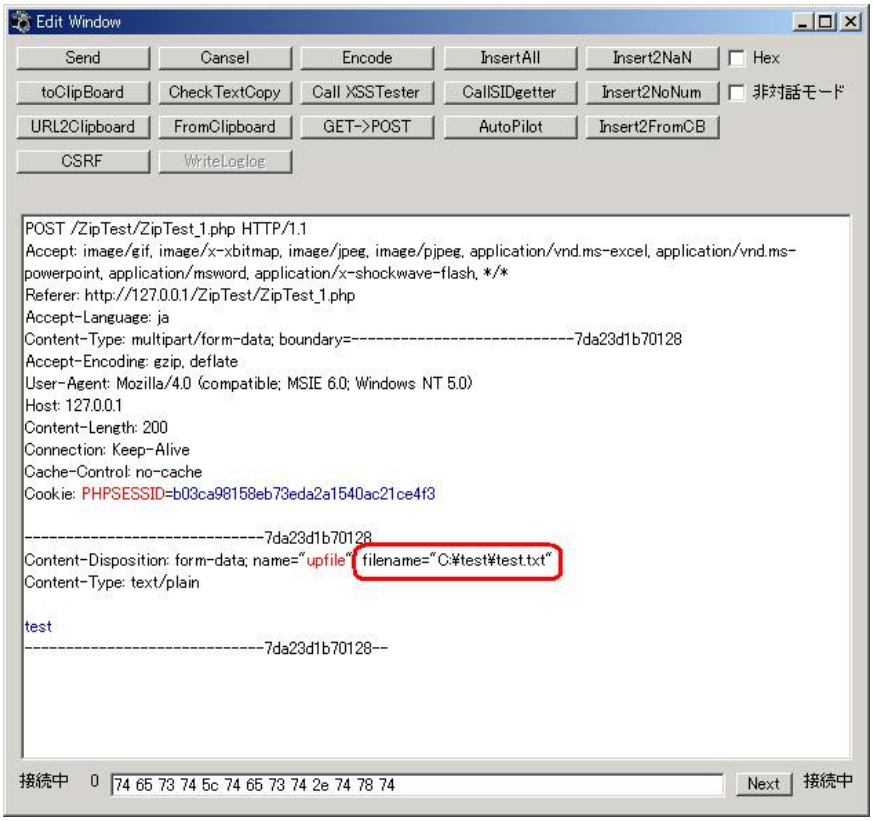

図 4.1-4 : 図 4.1-3のHTTPリクエスト

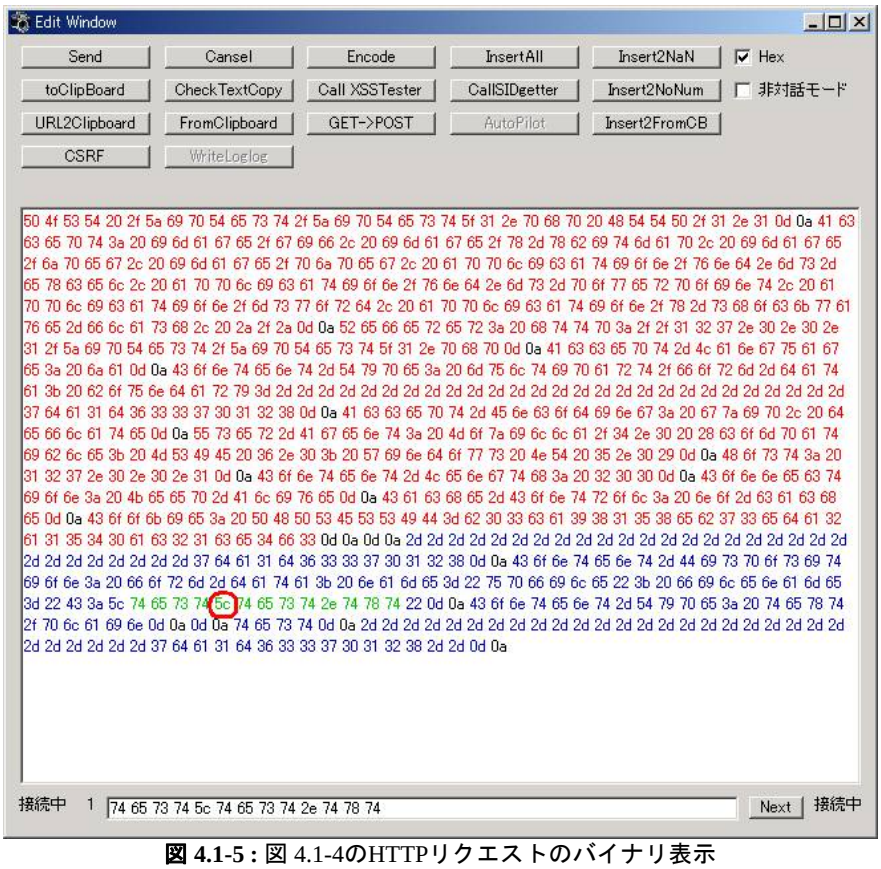

図 4.1-5 : 図 4.1-4のHTTPリクエストのバイナリ表示

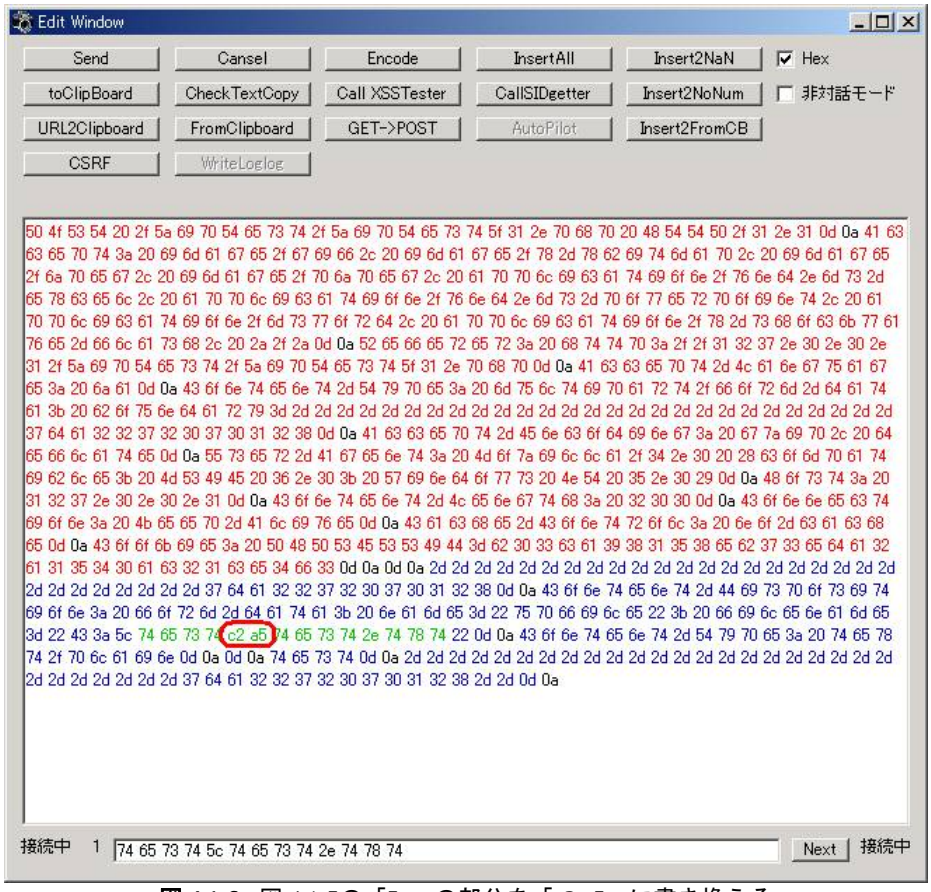

図 4.1-6 : 図 4.1-5の「5c」の部分を「c2 a5」に書き換える

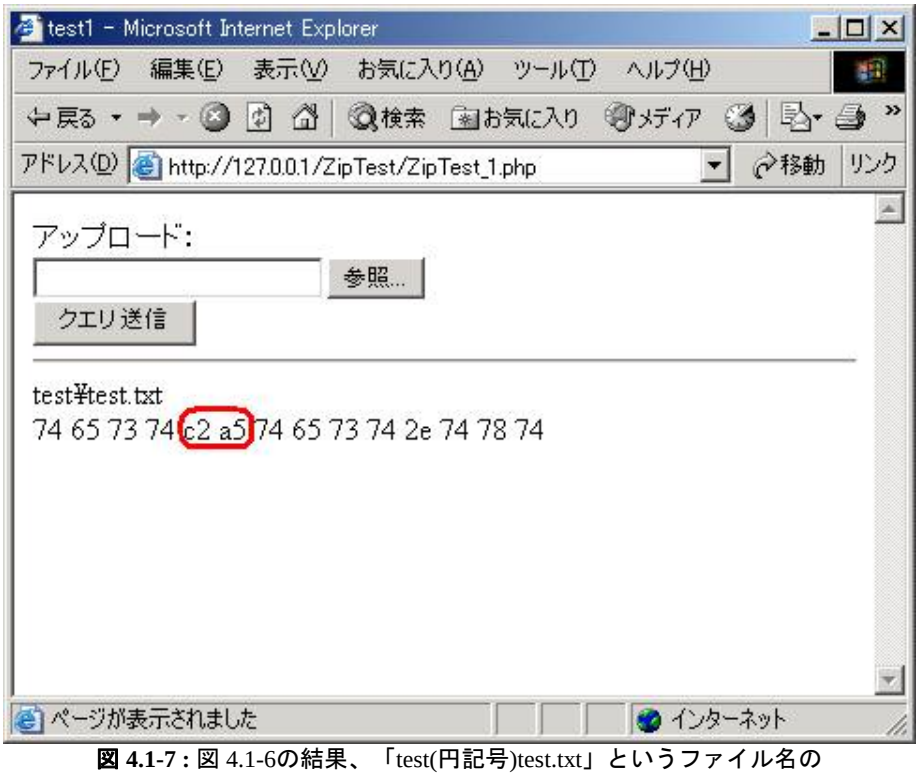

図 4.1-7 : 図 4.1-6の結果、「test(円記号)test.txt」というファイル名の
ファイルが圧縮されたことになっている

図 4.1-8 : 図 4.1-7の後の図 4.1-2には「test1.zip」というファイルが生成された

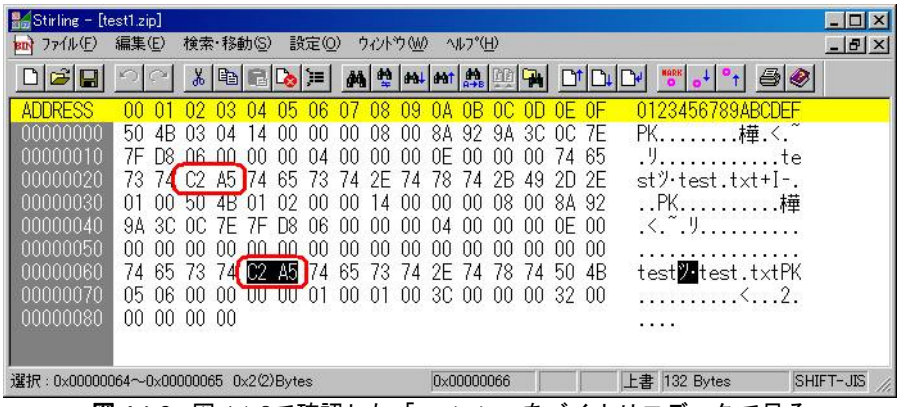

図 4.1-9 : 図 4.1-8で確認した「test1.zip」をバイナリエディタで見る

このように、zipファイル内の圧縮されたファイル名は、特に文字コードが変換されることなく、

そのままUTF-8 の文字コードで格納されている

4.2. zip.lib.php (phpMyAdmin) (Win32) の場合

検証環境

- MS-Windows2000 SP4
- Apache 2.2.11 for Win32
- PHP 5.2.13 for Win32
- phpMyAdmin 3.3.2

```
<html>
         <head>
          <meta http-equiv="content-type" content="text/html;charset=utf-8">
          <title>test2</title>
         </head>
         <body>
<?php
$display_data = "";
$zip_name = "test2.zip";
if(isset($zip_name)){

        if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
                $filename = $_FILES['upfile']['name'];
                $file_content = $_FILES['upfile']['tmp_name'];
                require_once('zip.lib.php');
                $zipfile = new zipfile();
                $handle = fopen($file_content, "rb");
                $contents = fread($handle, filesize($file_content));
                fclose($handle);
                $zipfile -> addFile( $contents, $filename );
                $zip_buffer = $zipfile->file();
                $handle = fopen($zip_name, "wb");
                fwrite($handle, $zip_buffer );
                fclose($handle);

                $hex_content = bin2hex($filename);
                $count = strlen($hex_content) / 2;

                for($i=0;$i<$count;$i++){
                        if($i==0){
                                $t=0;
                        }else{
                                $t=$i * 2;
                        }
                        $data = substr($hex_content , $t , 2);
                        $display_data = $display_data . $data . " ";
                }
        }
}

?>
                <form enctype="multipart/form-data" method="post" action="">
                アップロード：<br>
                <input type="file" name="upfile"><br>
                <input type="submit">
                </form>
                <hr>
                <?php echo $filename; ?><br>
                <?php echo $display_data; ?>
        </body>
</html>
```

図 4.2-1：検証コード

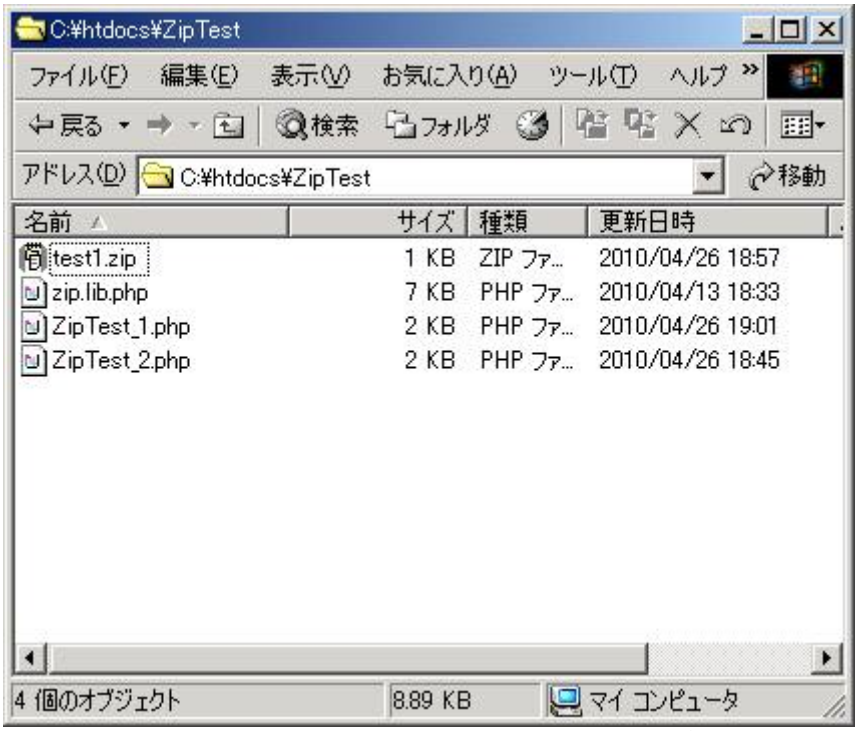

図 4.2-2 : 検証前の ZIP ファイルの保存先フォルダ

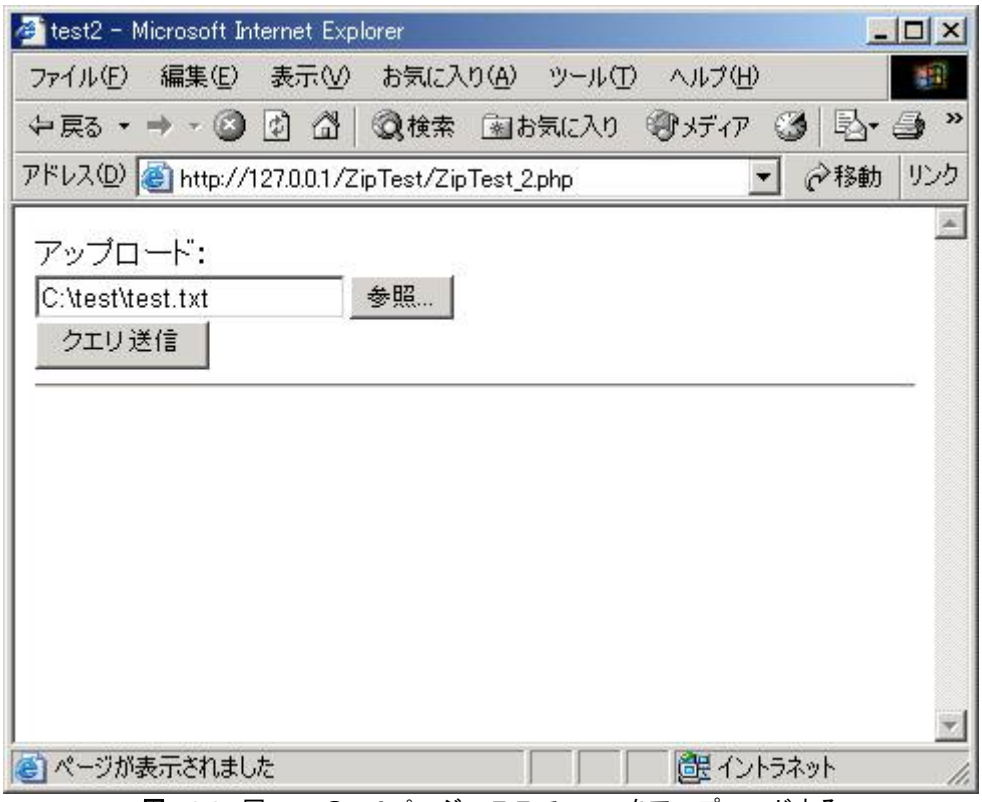

図 4.2-3 : 図 4.2-1のWebページ、ここでtest.txtをアップロードする

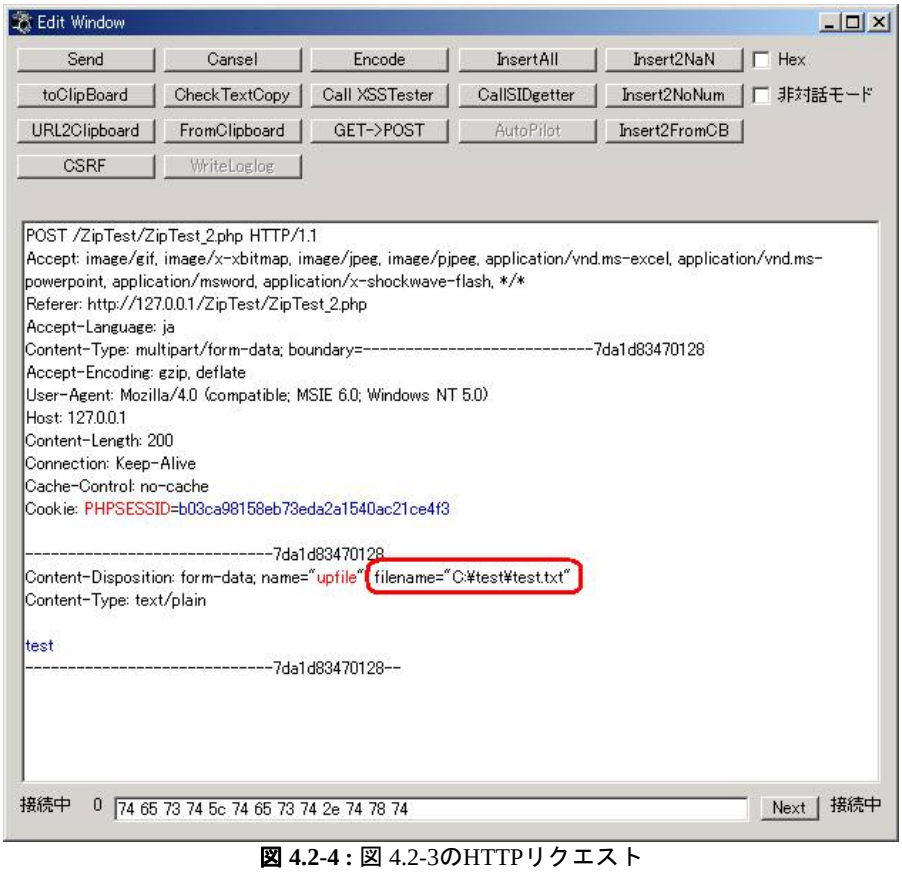

図 4.2-4 : 図 4.2-3のHTTPリクエスト

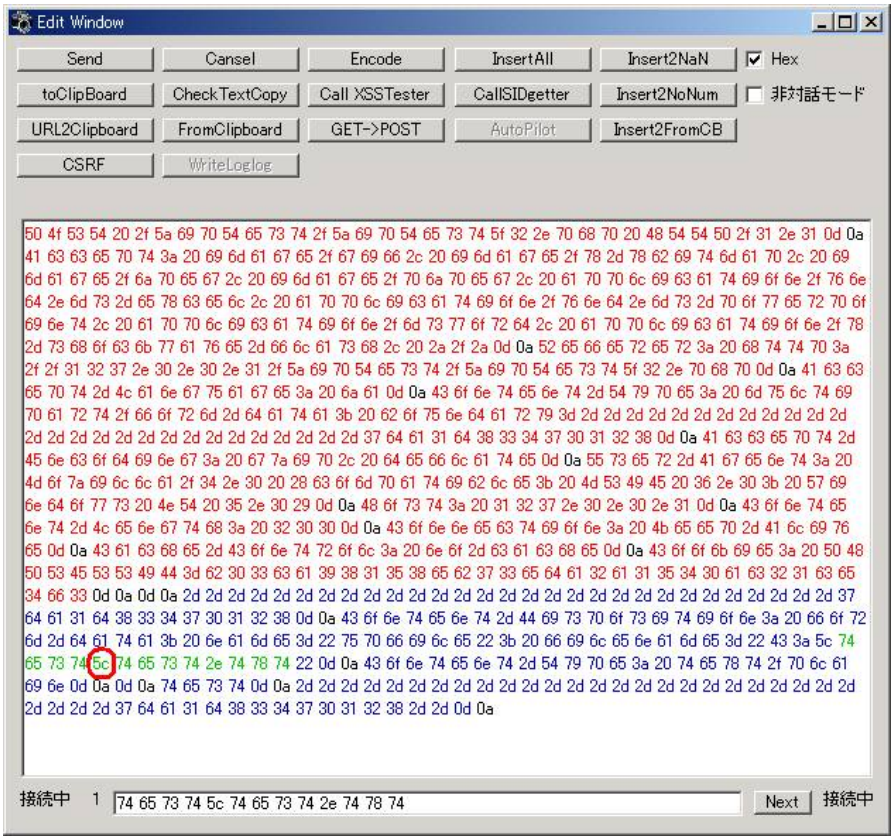

図 4.2-5 : 図 4.2-4のHTTPリクエストのバイナリ表示

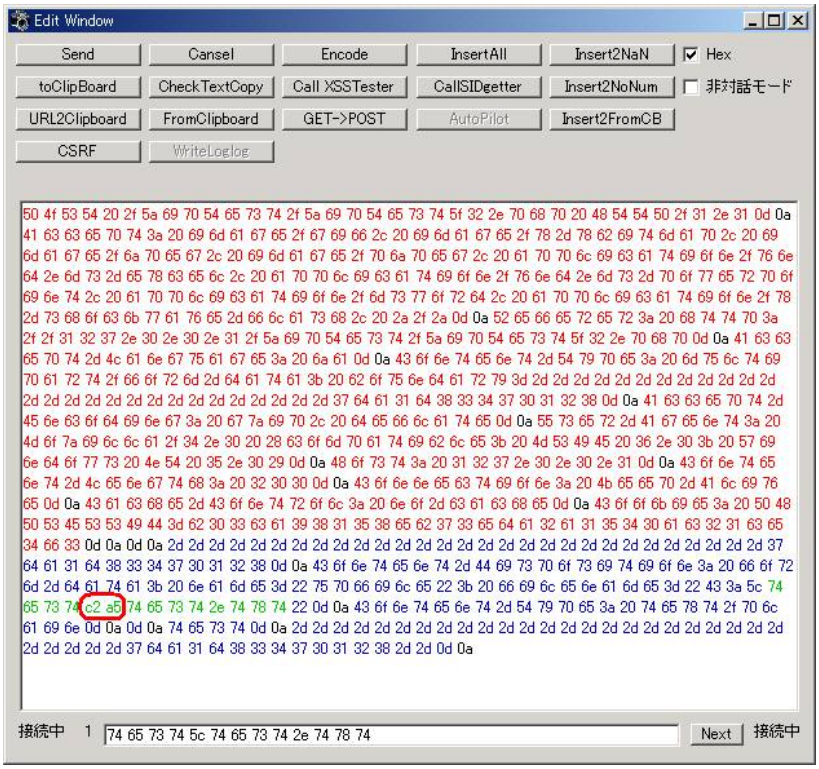

図 4.2-6 : 図 4.2-5の「5c」の部分を「c2 a5」に書き換える

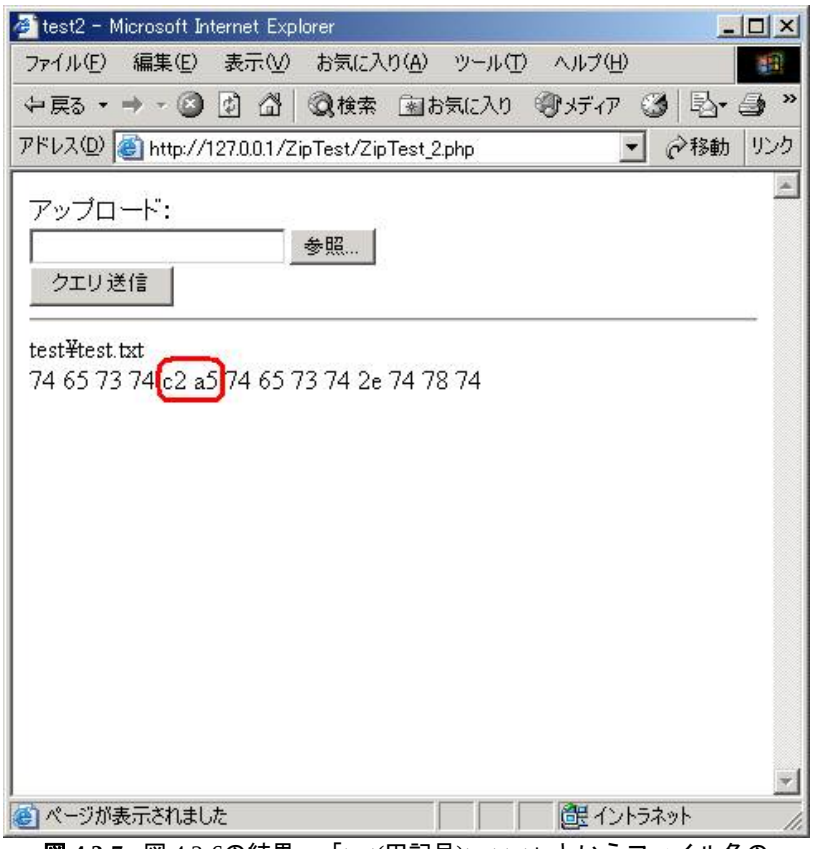

図 4.2-7 : 図 4.2-6の結果、「test(円記号)test.txt」というファイル名の
ファイルが圧縮されたことになっている

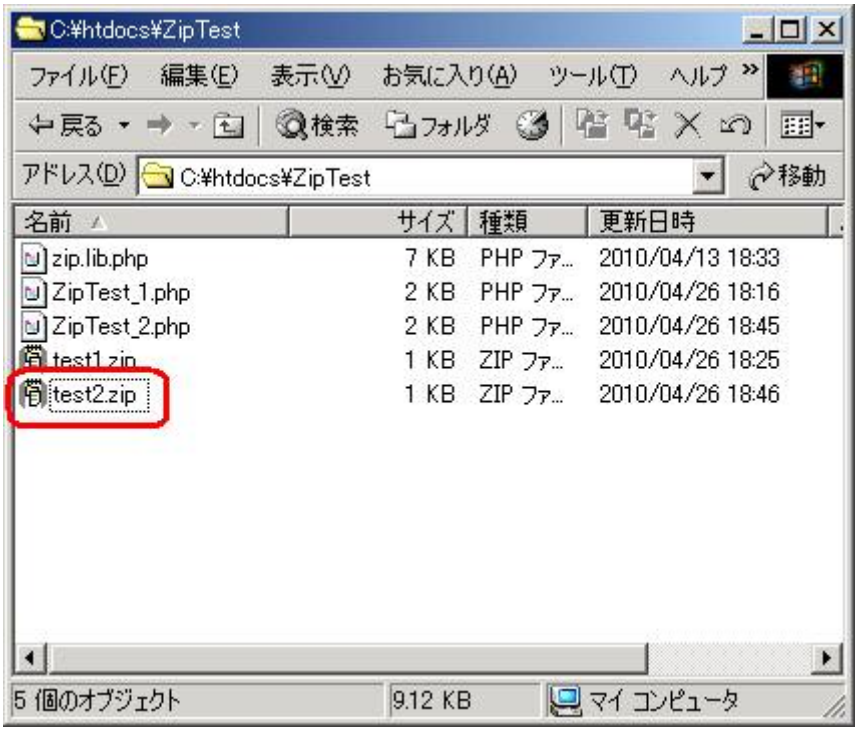

図 4.2-8 : 図 4.2-7の後の図 4.2-2には「test2.zip」というファイルが生成された

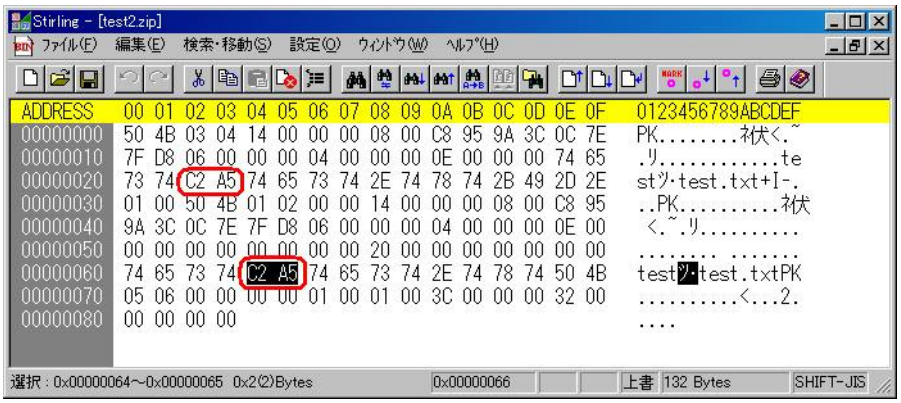

図 4.2-9 : 図 4.2-8で確認した「test2.zip」をバイナリエディタで見る

このように、zipファイル内の圧縮されたファイル名は、特に文字コードが変換されることなく、

そのままUTF-8 の文字コードで格納されている

4.3. php_zip.c (Linux) の場合

検証環境

- CentOS 5.3
- Apache 2.2.3
- PHP 5.3.2
- Zip version 1.9.1
- Libzip version 0.90

```
<html>
         <head>
          <meta http-equiv="content-type" content="text/html;charset=utf-8">
          <title>test1</title>
         </head>
         <body>
<?php
$display_data = "";
$zip_name = "/var/www/html/tmp/test1.zip";
if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
        $filename = $_FILES['upfile']['name'];
        $file_content = $_FILES['upfile']['tmp_name'];
        $contents = file_get_contents($file_content);
        $zip = new ZipArchive();
        $result = $zip->open($zip_name , ZipArchive::CREATE);

        if($result === TRUE){
                $zip->AddFromString($filename , $contents);
                $zip->close();
        }
        $hex_content = bin2hex($filename);
        $count = strlen($hex_content) / 2;

        for($i=0;$i<$count;$i++){
                if($i==0){
                        $t=0;
                }else{
                        $t=$i * 2;
                }
                $data = substr($hex_content , $t , 2);
                $display_data = $display_data . $data . " ";
        }
}

?>
          <form enctype="multipart/form-data" method="post" action="">
          アップロード：<br>
          <input type="file" name="upfile"><br>
          <input type="submit">
          <hr>
          <?php echo $filename; ?><br>
          <?php echo $display_data; ?>
         </body>
</html>
```

図 4.3-1：検証コード(図 4.1-1とほぼ同一である)

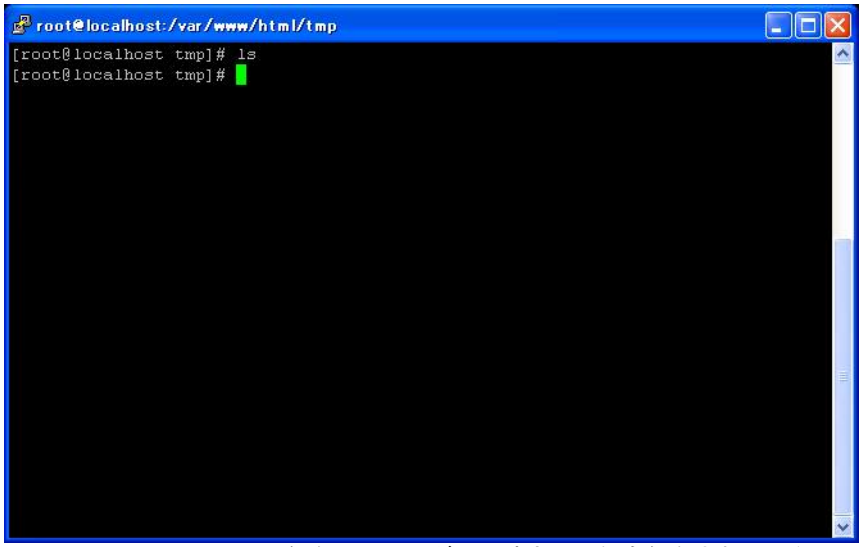

図 4.3-2 : ZIP ファイルの保存先フォルダ、現時点では何も保存されていない

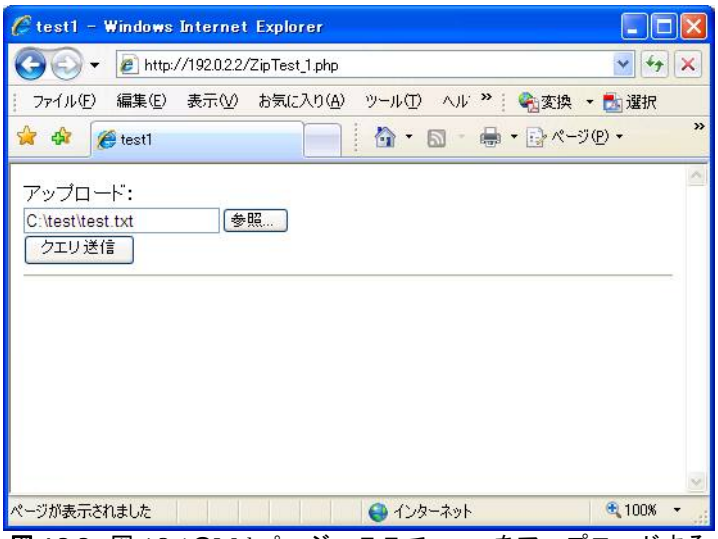

図 4.3-3 : 図 4.3-1のWebページ、ここでtest.txtをアップロードする

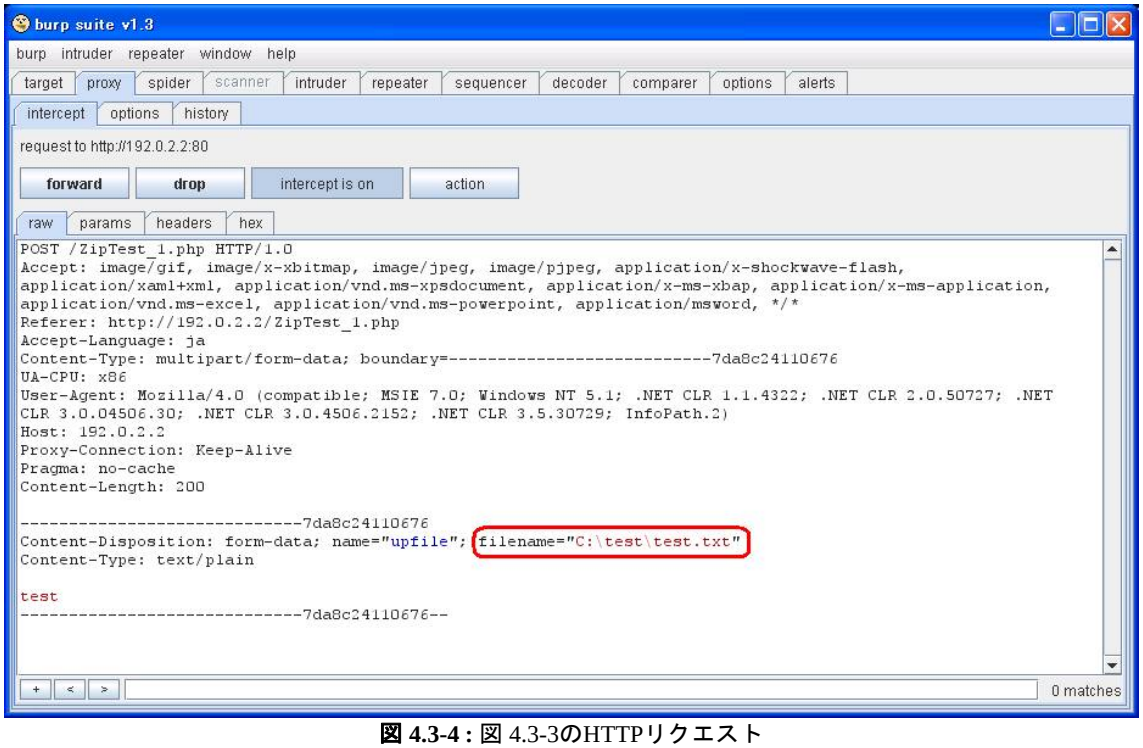

図 4.3-4 : 図 4.3-3のHTTPリクエスト

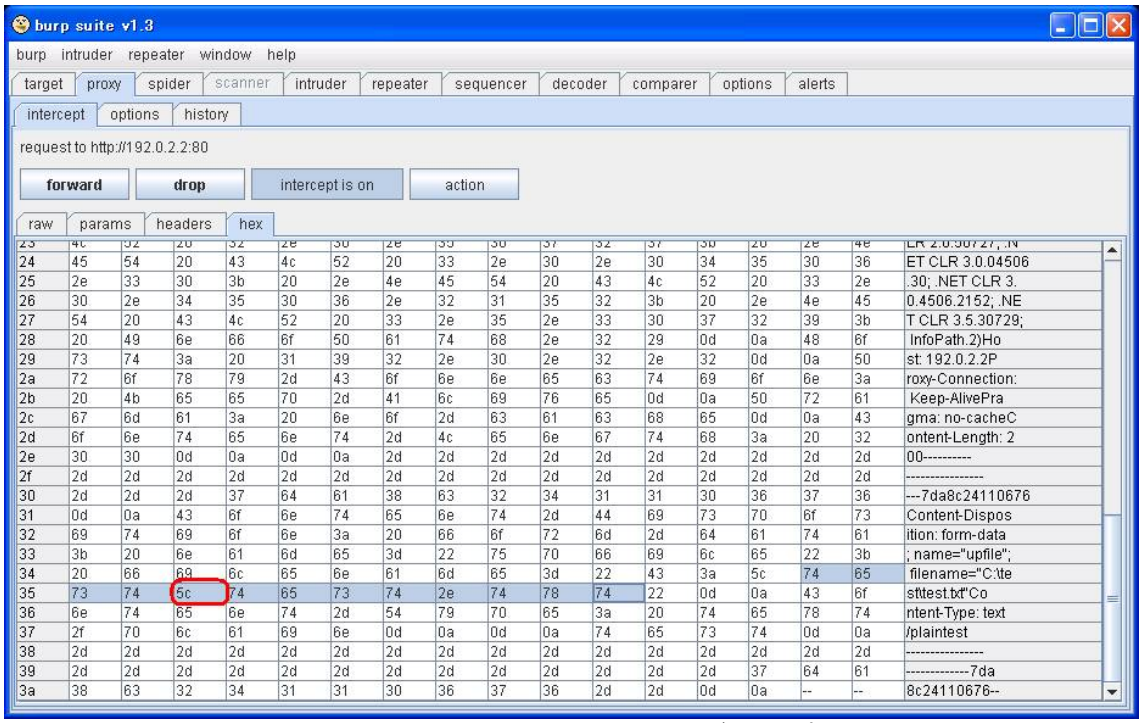

図 4.3-5 : 図 4.3-4のHTTPリクエストのバイナリ表示

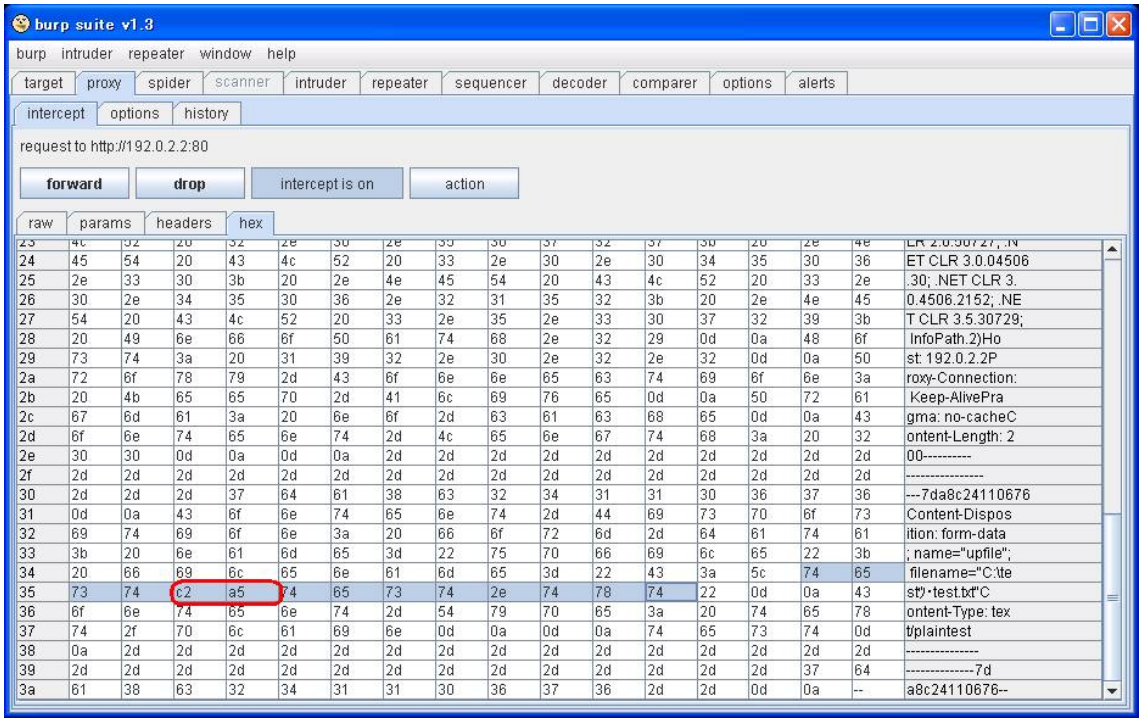

図 4.3-6 : 図 4.3-5の「5c」の部分を「c2 a5」に書き換える

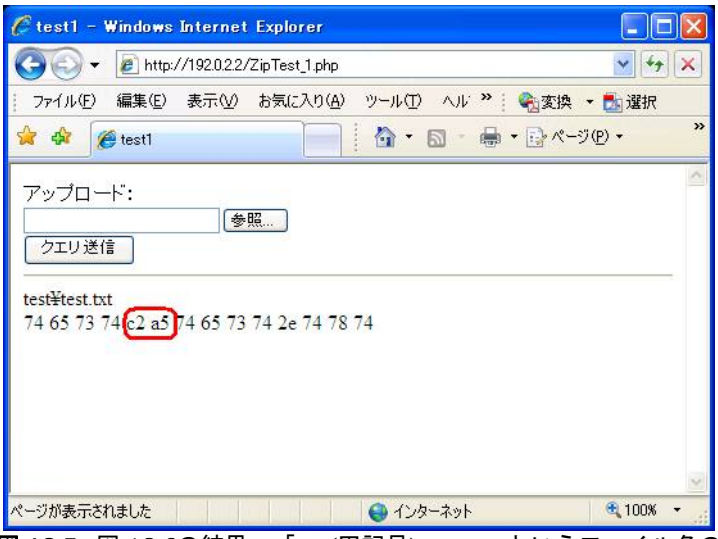

図 4.3-7 : 図 4.3-6の結果、「test(円記号)test.txt」というファイル名のファイルが圧縮されたことになっている

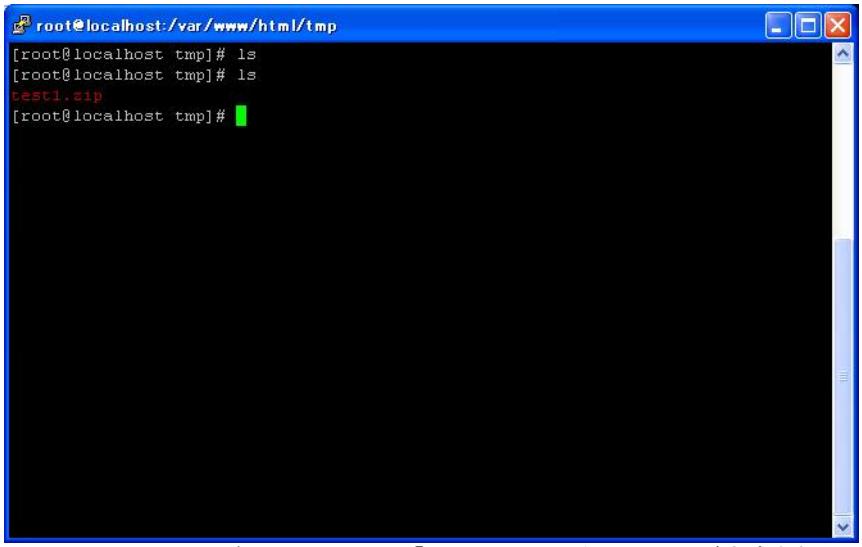

図 4.3-8 : 図 4.3-7の後の図 4.3-2には「test1.zip」というファイルが生成された

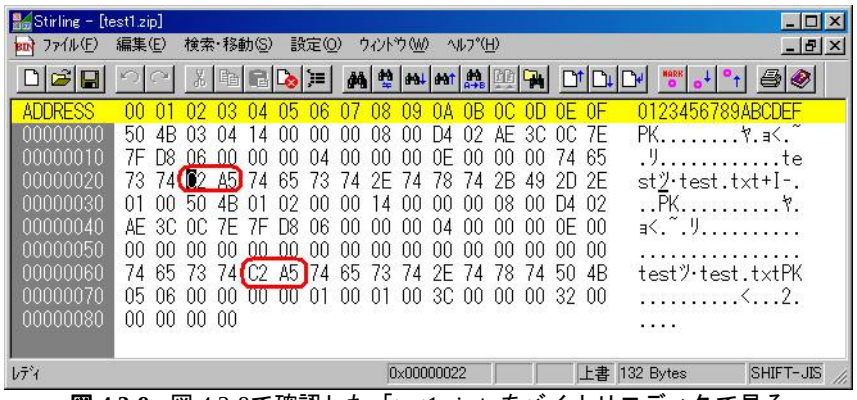

図 4.3-9 : 図 4.3-8で確認した「test1.zip」をバイナリエディタで見る

このように、zipファイル内の圧縮されたファイル名は、特に文字コードが変換されることなく、

そのままUTF-8 の文字コードで格納されている

4.4. zip.lib.php (phpMyAdmin) (Linux) の場合

検証環境

- CentOS 5.3
- Apache 2.2.3
- PHP 5.3.2
- phpMyAdmin 3.3.2

```
<html>
         <head>
          <meta http-equiv="content-type" content="text/html;charset=utf-8">
          <title>test2</title>
         </head>
         <body>
<?php
$display_data = "";
$zip_name = "/var/www/html/tmp/test2.zip";
if(isset($zip_name)){

        if(is_uploaded_file($_FILES['upfile']['tmp_name'])){
                $filename = $_FILES['upfile']['name'];
                $file_content = $_FILES['upfile']['tmp_name'];
        require_once('zip.lib.php');
                $zipfile = new zipfile();
                $handle = fopen($file_content, "rb");
                $contents = fread($handle, filesize($file_content));
                fclose($handle);
                $zipfile -> addFile( $contents, $filename );
                $zip_buffer = $zipfile->file();
                $handle = fopen($zip_name , "wb");
                fwrite($handle, $zip_buffer );
                fclose($handle);

                $hex_content = bin2hex($filename);
                $count = strlen($hex_content) / 2;

                for($i=0;$i<$count;$i++){
                        if($i==0){
                                $t=0;
                        }else{
                                $t=$i * 2;
                        }
                        $data = substr($hex_content , $t , 2);
                        $display_data = $display_data . $data . " ";
                }
        }
}

?>
                        <form enctype="multipart/form-data" method="post"
action="">
                                アップロード：<br>
                                <input type="file" name="upfile"><br>
                                <input type="submit">
                                <hr>
                                <?php echo $filename; ?><br>
                                <?php echo $display_data; ?>
                </body>

</html>
```

図 4.4-1：検証コード(図 4.2-1とほぼ同一である)

図 4.4-2 : 図 4.4-1のWebページ、ここでtest.txtをアップロードする

検証前のZIPファイルの保存先フォルダの状態は、図 4.3-8である

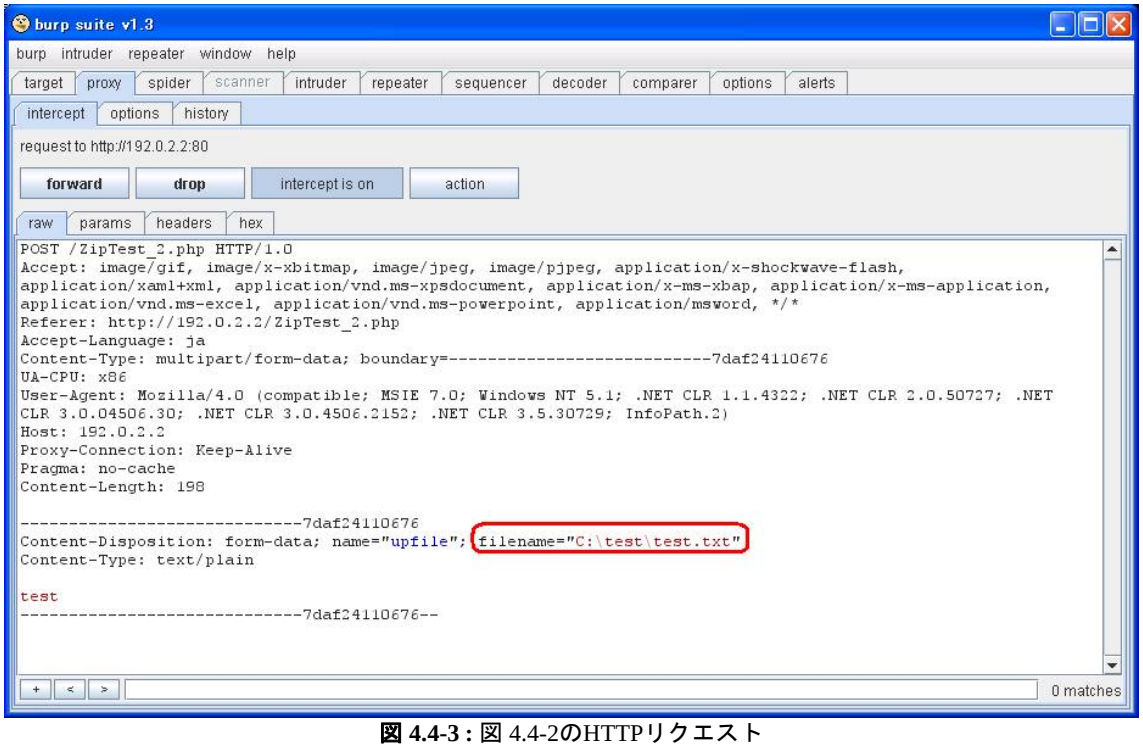

図 4.4-3 : 図 4.4-2のHTTPリクエスト

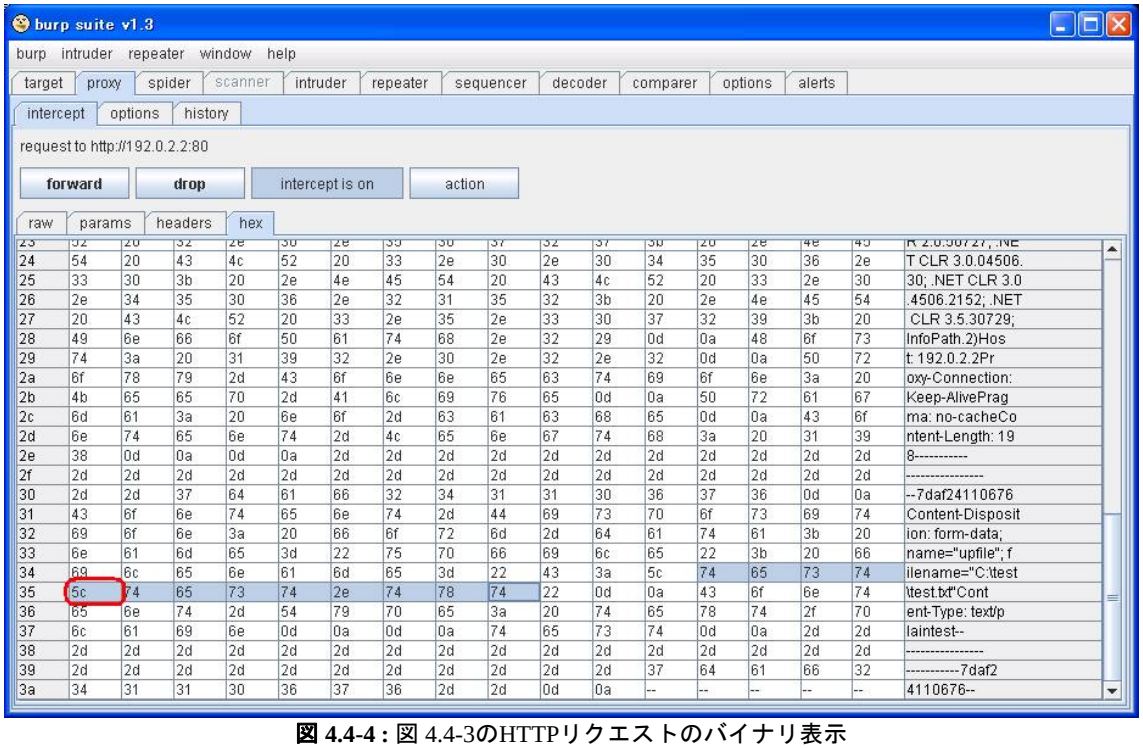

図 4.4-4 : 図 4.4-3のHTTPリクエストのバイナリ表示

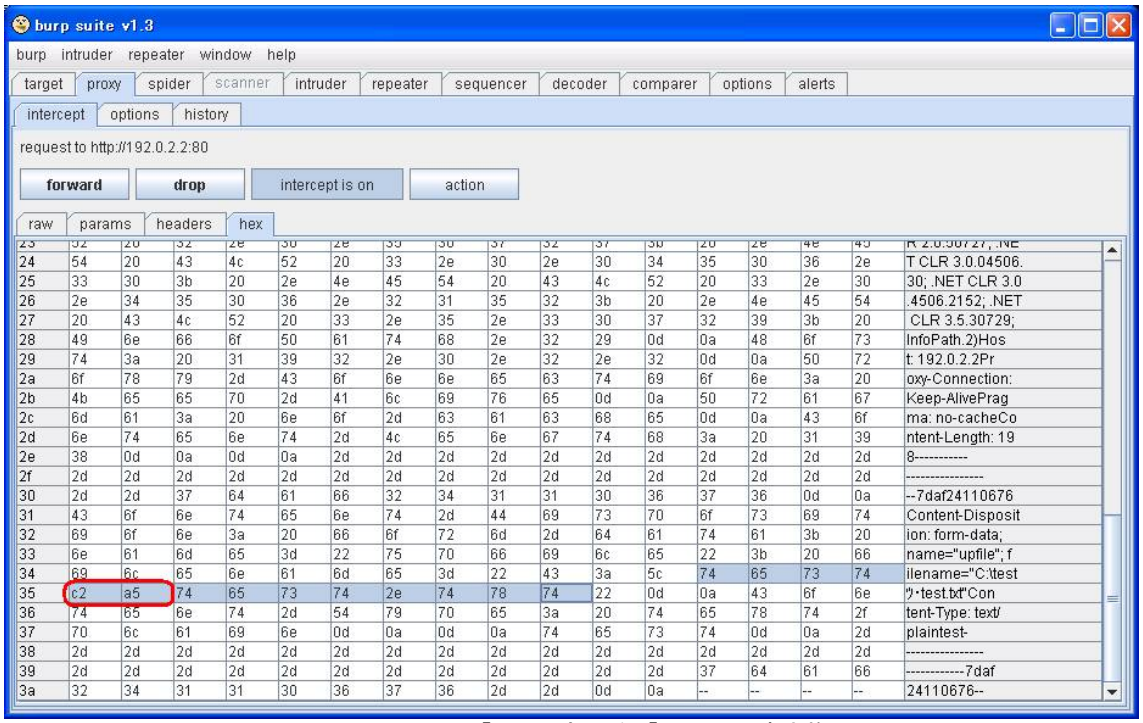

図 4.4-5 : 図 4.4-4の「5c」の部分を「c2 a5」に書き換える

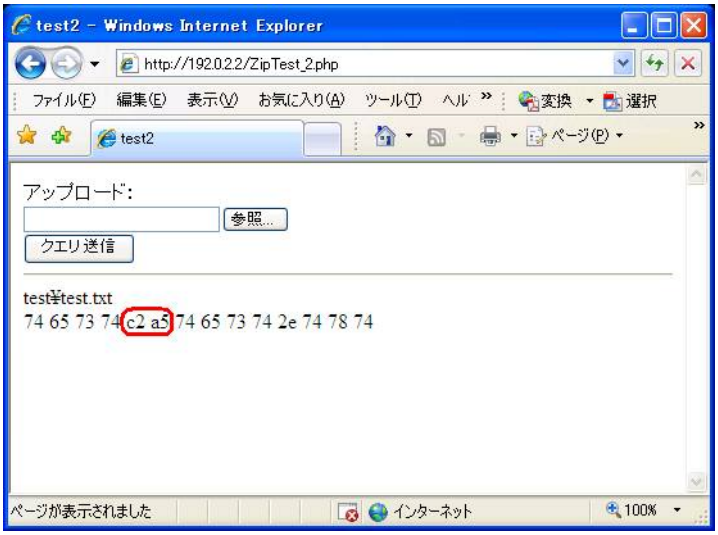

図 4.4-6 : 図 4.4-5の結果、「test(円記号)test.txt」というファイル名の
ファイルが圧縮されたことになっている

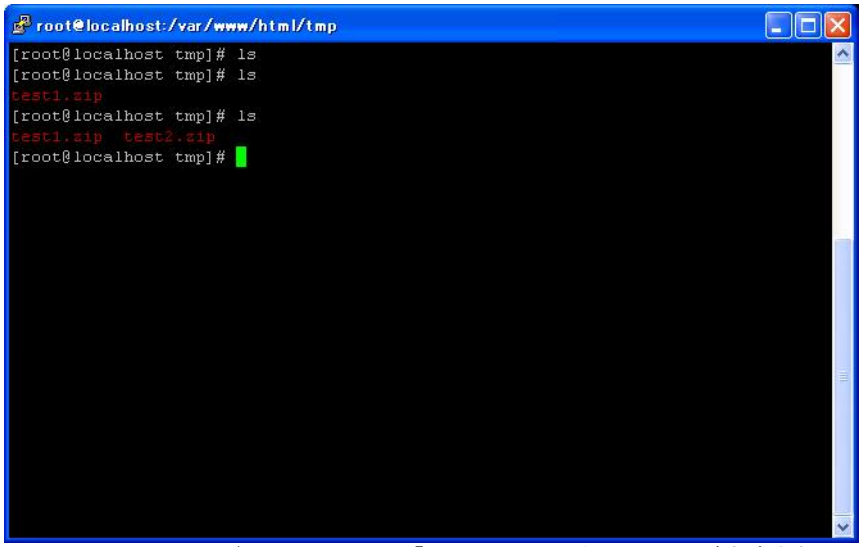

図 4.4-7 : 図 4.4-6の後の図 4.3-8には「test2.zip」というファイルが生成された

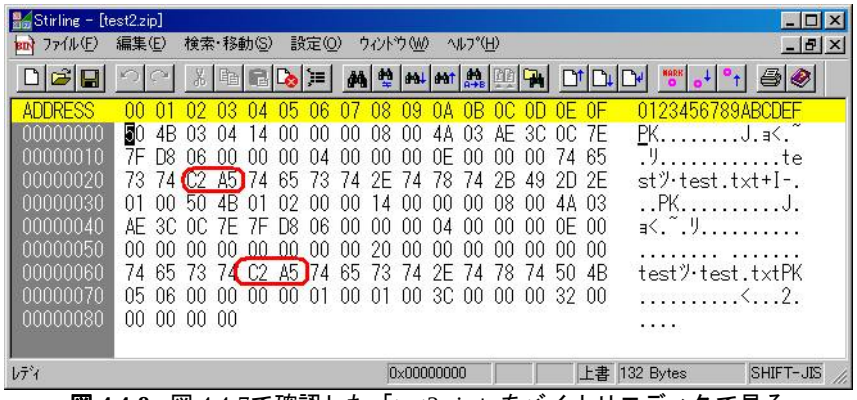

図 4.4-8 : 図 4.4-7で確認した「test2.zip」をバイナリエディタで見る

このように、zipファイル内の圧縮されたファイル名は、特に文字コードが変換されることなく、
そのままUTF-8 の文字コードで格納されている

4.5. zip.lib.php (phpMyAdmin) (Linux) の場合その 2

検証環境

- CentOS 5.3
- Apache 2.2.3
- PHP 5.3.2
- phpMyAdmin 3.3.2

本項では、ファイル名を UTF-8 で受け取りながら、PHP スクリプト上で ShiftJIS に変換してみる。この ShiftJIS への変換によって、作成された ZIP ファイルを MS-Windows 上で展開すると、文字化けは発生しない。
しかしながら、PHP スクリプト上で、ディレクトリ・トラバーサルに関するバリデーションを実施しないと、展開時に、ディレクトリ・トラバーサル脆弱性が発現してしまうだろう。

図 4.5-1 : で確認した「test2.zip」をバイナリエディタで見る
このように、zip ファイル内の圧縮されたファイル名は、特に文字コードが変換されることなく、
そのまま UTF-8 の文字コードで格納されている

5. zipコマンドの場合

最近のLinux系では、既定の文字コードがUTF-8 の場合が多いのではないだろうか。そのような状況の場合、ファイル名に「円記号(u00a5)」を含ませることができ、このファイルを圧縮する際の挙動を確認した。CGIプログラムから呼び出すことができれば、Webアプリケーションとして機能するはずだからである[1]。

[1] 実際に、Web アプリケーション／CGI プログラムから OS コマンドを呼び出す場面とは、それほど多くないと思われる。

5.1. そのまま「\」を指定する場合

Linux 上のファイルシステムで「\(0x5c:バックスラッシュ)」は特別な意味はない。よって、そのまま与えたらどうなるか確認してみる。
図 5.1-2を見るまでもなく、当然の結果として、圧縮ファイル内のファイル名に「\」が含まれており、この圧縮ファイルをWindows上の古い展開ツールで展開すると、思わぬディレクトリ上にファイルが展開される危険性がある。
しかしながら、ほとんどのプログラマは、ファイル名に「\」が含まれていないことぐらいは確認していると思われるため、この項目がセキュリティ脆弱性として発現する機会は非常に稀であるだろう。

検証環境

- CentOS 5.1
- zip コマンド 2.31

またそのような場合は、OS コマンドインジェクション対策も行う必要があるかもしれない。

```
# ls -alF

合計 12
drwxr-xr-x  2 root root 4096  4月 27 13:01 ./
drwxrwxrwt 15 root root 4096  4月 27 13:01 ../
# zip

Copyright (C) 1990-2005 Info-ZIP
Type 'zip "-L"' for software license.
Zip 2.31 (March 8th 2005). Usage:
zip [-options] [-b path] [-t mmddyyyy] [-n suffixes] [zipfile list] [-xi list]
  The default action is to add or replace zipfile entries from list, which
  can include the special name - to compress standard input.
  If zipfile and list are omitted, zip compresses stdin to stdout.
  -f   freshen: only changed files  -u   update: only changed or new files
  -d   delete entries in zipfile    -m   move into zipfile (delete files)
  -r   recurse into directories     -j   junk (don't record) directory names
  -0   store only                   -l   convert LF to CR LF (-ll CR LF to LF)
  -1   compress faster              -9   compress better
  -q   quiet operation              -v   verbose operation/print version info
  -c   add one-line comments        -z   add zipfile comment
  -@   read names from stdin        -o   make zipfile as old as latest entry
  -x   exclude the following names  -i   include only the following names
  -F   fix zipfile (-FF try harder) -D   do not add directory entries
  -A   adjust self-extracting exe   -J   junk zipfile prefix (unzipsfx)
  -T   test zipfile integrity       -X   eXclude eXtra file attributes
  -y   store symbolic links as the link instead of the referenced file
  -R   PKZIP recursion (see manual)
  -e   encrypt                      -n   don't compress these suffixes
# echo Hello > abc¥¥xyz.txt

# zip test.zip *.txt

  adding: abc¥xyz.txt (stored 0%)
# ls -alF

合計 20
drwxr-xr-x  2 root root 4096  4月 27 13:02 ./
drwxrwxrwt 15 root root 4096  4月 27 13:01 ../
-rw-r--r--  1 root root    6  4月 27 13:02 abc¥xyz.txt
-rw-r--r--  1 root root  160  4月 27 13:02 test.zip
```

図 5.1-1：検証結果

「abc\xyz.txt」というファイルを「test.zip」に圧縮した

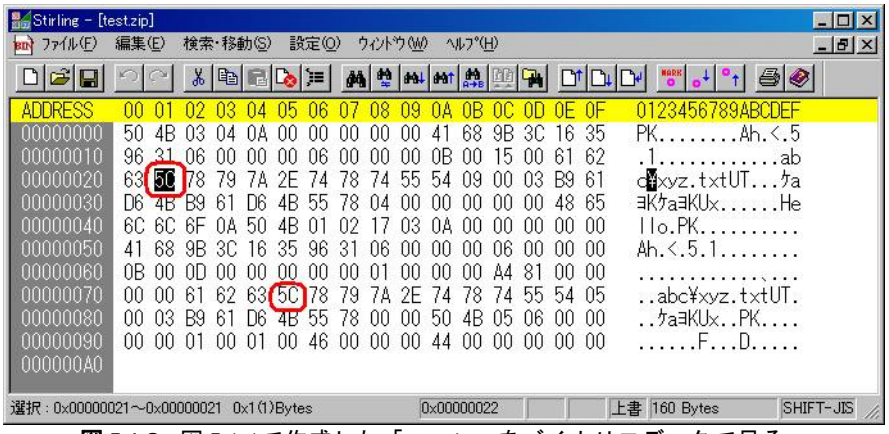

図 5.1-2 : 図 5.1-1で作成した「test.zip」をバイナリエディタで見る。

当然であるが「0x5c」で指定されたファイル名はそのまま埋め込まれている。

5.2. UTF-8 で「円記号(u00a5)」を指定する場合

次は、「LANG=UTF-8」の Linux 上のファイルシステムで「円記号(u00a5)」を含むファイル名の場合について確認してみる。
図 5.2-2のように、別の文字コードに変換されることなく圧縮されているため、本文書のようなサニタイズ回避テクニックは有効に働かないだろう。

検証環境

- CentOS 5.1
- zip コマンド 2.31

```
# env | grep "LANG"
LANG=ja_JP.UTF-8
# ls -alF

合計 16
drwxr-xr-x  2 root root 4096  4 月 27 13:52 ./
drwxrwxrwt 13 root root 4096  4 月 27 13:48 ../
-rw-r--r--  1 root root  126  4 月 27 13:50 a.pl
# cat a.pl

#! /usr/local/bin/perl

$str = ">abc¥xc2¥xa5xyz.txt";
open FILE,$str;
print FILE "Hello";
close(FILE);
print "END¥n";
__END__
# perl a.pl
END
# zip test.zip *.txt

  adding: abc¥xyz.txt (stored 0%)
# ls -alF

合計 24
drwxr-xr-x  2 root root 4096  4 月 27 13:52 ./
drwxrwxrwt 13 root root 4096  4 月 27 13:48 ../
-rw-r--r--  1 root root  126  4 月 27 13:50 a.pl
-rw-r--r--  1 root root    5  4 月 27 13:52 abc¥xyz.txt
-rw-r--r--  1 root root  161  4 月 27 13:52 test.zip
```

図 5.2-1：検証結果

「abc(円記号)xyz.txt」というファイルを perl で作成し「test.zip」に圧縮した

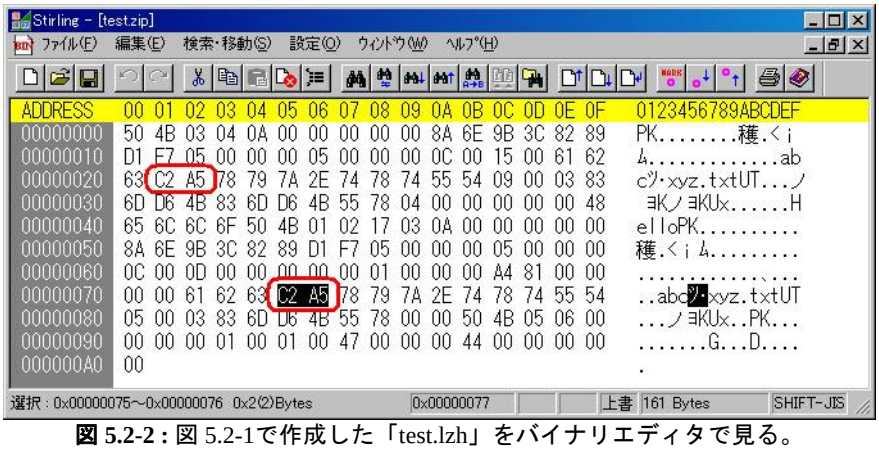

図 5.2-2：図 5.2-1で作成した「test.lzh」をバイナリエディタで見る。

ファイル名がUTF-8(UNICODE)のまま保存されているのが確認できる。

UNICODEで保存されているため、本文書のサニタイズ回避テクニックはうまく行かないだろう

6. UNLHA32.DLL の場合

Windows 上であれば、ファイルシステム NTFS 上に UNICODE でファイルを作成することが可能である。よって、ファイル名に円記号(u00a5)を含ませることができ、このファイルを圧縮する際の挙動を確認した。CGI プログラムから呼び出すことができれば、Web アプリケーションとして機能するはずだからである。

6.1. Lha32.exe の場合

検証環境

- MS-Windows XP SP3
- Lha32.exe 1.06
- UNLHA32.DLL 2.63

http://www.vector.co.jp/soft/win95/util/se028209.html で配布されている UNLHA32.DLL をコマンドラインから呼び出すツールである。
このコマンドを例に、UNLHA32.DLL の挙動について確認した。

```
C:¥z>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は 24AC-3307 です

 C:¥z のディレクトリ

2010/04/27  11:41    <DIR>          .
2010/04/27  11:41    <DIR>          ..
2010/04/27  11:38                 7 abc.txt
2010/04/27  11:38                 7 abc¥xyz.txt
               2 個のファイル                  14 バイト
               2 個のディレクトリ    9,300,709,376 バイトの空き領域

C:¥z>lha32
Lha32 version 1.06                    Copyright (c) Take, 1995-1996
=== <<< A High-Performance File-Compression Program >>> ========  96/10/26  ===
Usage: Lha32 <command> [/option[-+012|WDIR]] <archive[.LZH]> [DIR¥] [filenames]
--------------------------------------------------------------------------------
  <command>
     a: Add files           u: Update files        m: Move files
     f: Freshen files       d: Delete files        p: disPlay files
     e: Extract files       x: eXtract files with pathnames
     l: List of files       v: View listing of files with pathnames
     s: make a Self-extracting archive   t: Test the integrity of an archive
  <option>
     r: Recursively collect files        w: assign Work directory
     x: allow eXtended file names        m: no Message for query
     p: distinguish full Path names      c: skip time-stamp Check
     a: allow any Attributes of files    z: Zero compression (only store)
```

```
     t: archive's Time-stamp option      h: select Header level (default = 2)
     o: use Old compatible method        n: display No indicator a/o pathname
     i: not Ignore lower case            l: display Long name with indicator
     s: Skip by time is not reported     -: '@' and/or '-' as usual letters
================================================================================
 You may copy or distribute this software free of charge.
                                                      gi8s-tkuc@asahi-net.or.jp
UNLHA32.DLL Version 2.63                                  Nifty-Serve  QZI12273
C:¥z>lha32 a c:¥z¥test.lzh *.txt

Creating archive : c:/z/test.lzh

Frozen    ==> 100% abc.txt
Frozen    ==> 100% abc_xyz.txt

C:¥z>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は 24AC-3307 です

 C:¥z のディレクトリ

2010/04/27  11:54    <DIR>          .
2010/04/27  11:54    <DIR>          ..
2010/04/27  11:38                 7 abc.txt
2010/04/27  11:38                 7 abc¥xyz.txt
2010/04/27  11:54               196 test.lzh
               3 個のファイル                 210 バイト
               2 個のディレクトリ    9,300,705,280 バイトの空き領域

C:¥z>
```

図 6.1-1：検証結果

「abc(円記号)xyz.txt」というファイルを「test.lzh」に圧縮した

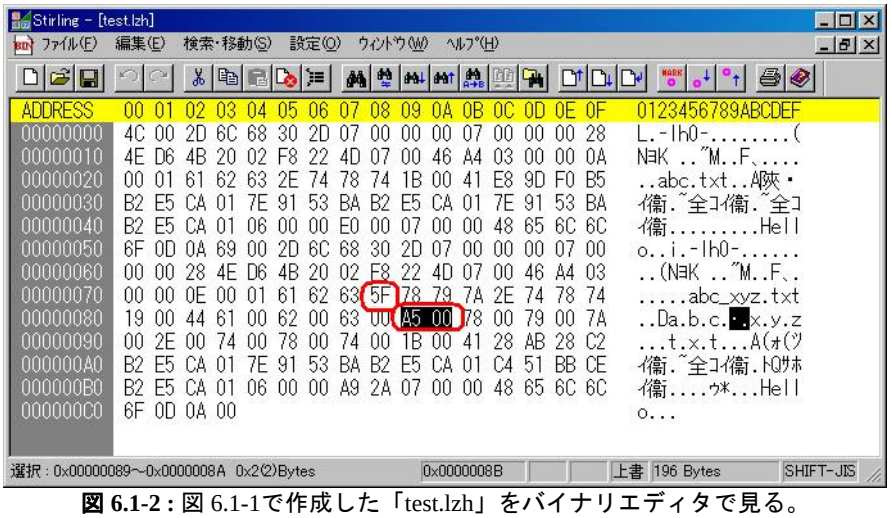

図 6.1-2：図 6.1-1で作成した「test.lzh」をバイナリエディタで見る。

ファイル名がUTF-16(UNICODE)と「_(アンダーバー)」に変換されて保存されているのが確認できる。

UNICODEで保存されているため、本文書のサニタイズ回避テクニックはうまく行かないだろう。

```
C:¥z>md a

C:¥z>cd a

C:¥z¥a>copy ..¥test.lzh .
        1 個のファイルをコピーしました。

C:¥z¥a>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は 24AC-3307 です

 C:¥z¥a のディレクトリ

2010/04/27  12:10    <DIR>          .
2010/04/27  12:10    <DIR>          ..
2010/04/27  11:54               196 test.lzh
               1 個のファイル                 196 バイト
               2 個のディレクトリ    9,298,309,120 バイトの空き領域

C:¥z¥a>lha32 x test.lzh

Extracting from archive : C:/z/a/test.lzh

Melted   abc.txt
Melted   abc_xyz.txt


C:¥z¥a>dir
 ドライブ C のボリューム ラベルがありません。
 ボリューム シリアル番号は 24AC-3307 です

 C:¥z¥a のディレクトリ

2010/04/27  12:10    <DIR>          .
2010/04/27  12:10    <DIR>          ..
2010/04/27  11:38                 7 abc.txt
2010/04/27  11:38                 7 abc¥xyz.txt
2010/04/27  11:54               196 test.lzh
               3 個のファイル                 210 バイト
               2 個のディレクトリ    9,298,300,928 バイトの空き領域

C:¥z¥a>
```

図 6.1-3 : 図 6.1-1で作成した「test.lzh」を実際に展開した結果。

UNICODEのファイル名はそのままUNICODEのファイル名として展開されているため、

特にセキュリティ上の問題を誘発していない。

図 6.1-4 : 図 6.1-1で作成した「test.lzh」をeo1.5.2 で展開した結果。

UNICODEで与えた円記号(u00a5)は「_(アンダーバー)」に置換されている。

7. ASP(VBScript) の場合

7.1. Shell.Applicationオブジェクト の場合

検証環境
- MS-Windows XP SP3
- WSH 5.7

WindowsXP 以降では、Shell.Application オブジェクト(COM/ActiveX)を使うことで、
ASP(VBScript)でも、ZIP ファイルを作成することができる。
WSH(VBScript)での挙動を確認した。

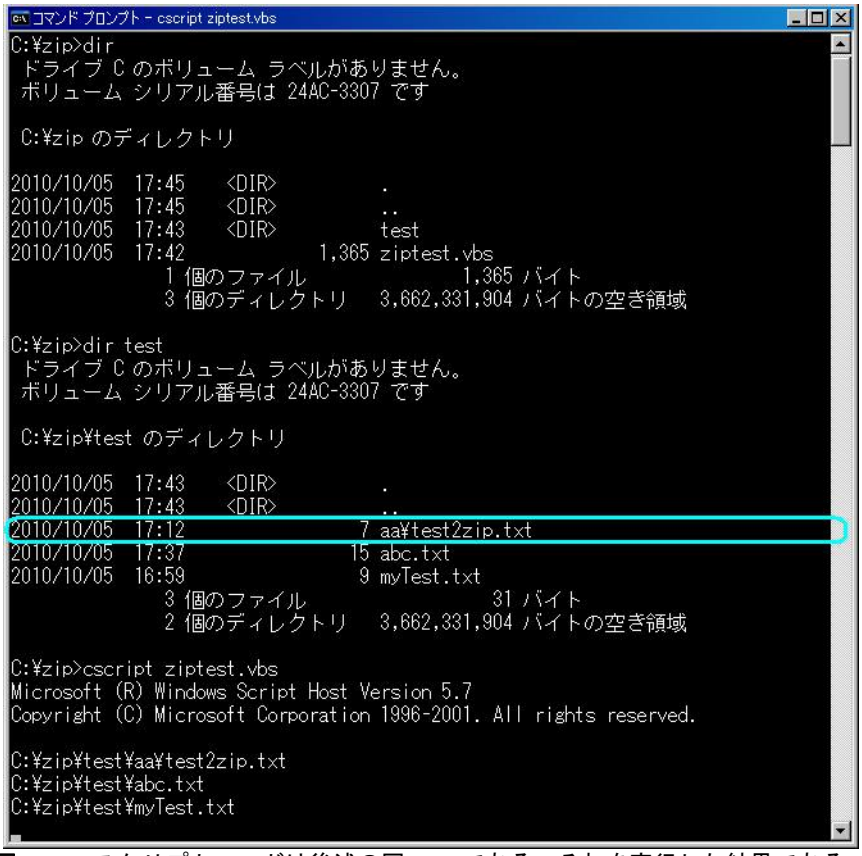

図 7.1-1 : スクリプトコードは後述の図 7.1-7である。それを実行した結果である。

図 7.1-2のエラーを表示したが、プロンプトは返ってこない。DoS攻撃が可能になるかもしれない。

(最後の処理で無限ループになっている可能性がある)

図 7.1-2 : 図 7.1-1時のエラーメッセージ。

UNICODEの円記号を拒絶する内容だ。

図 7.1-3 : 図 7.1-1後、[CTRL]+[C]で強制終了してみると、ZIPファイルは完成していた。

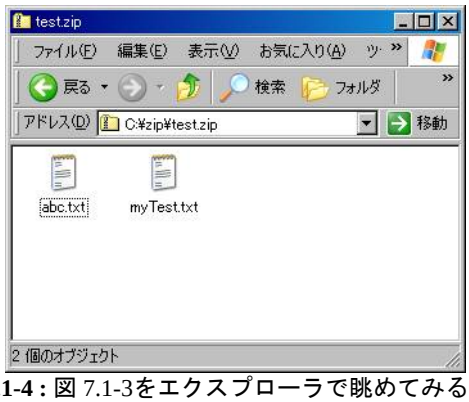

図 7.1-4 : 図 7.1-3をエクスプローラで眺めてみると、

UNICODEの円記号がファイル名に含むファイルは含まれていなかった。

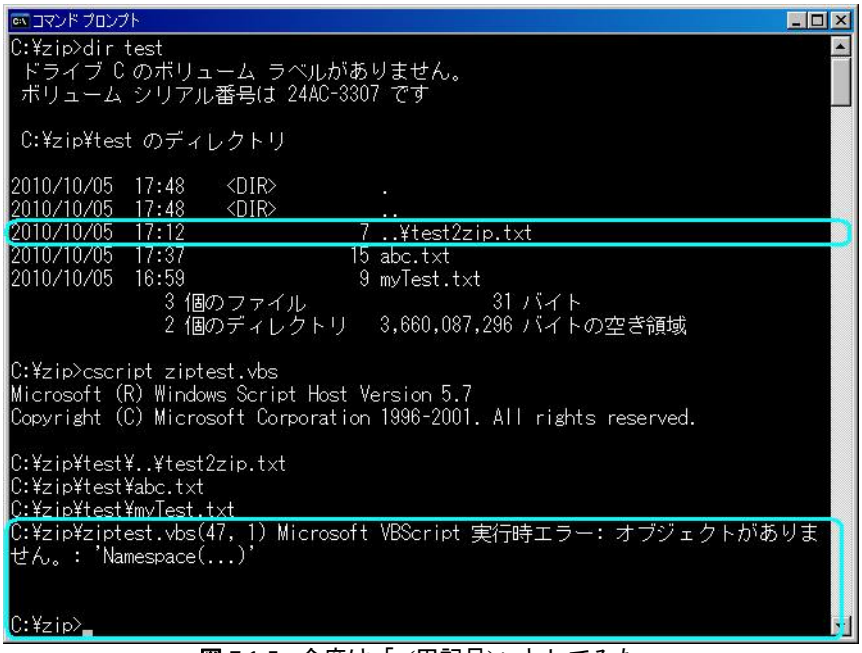

図 7.1-5 : 今度は「..(円記号)」としてみた。

今度は、CUI 上にエラーが表示され、プロンプトが戻ってきている。

(なぜ最後の処理で無限ループにならないのだろう?)

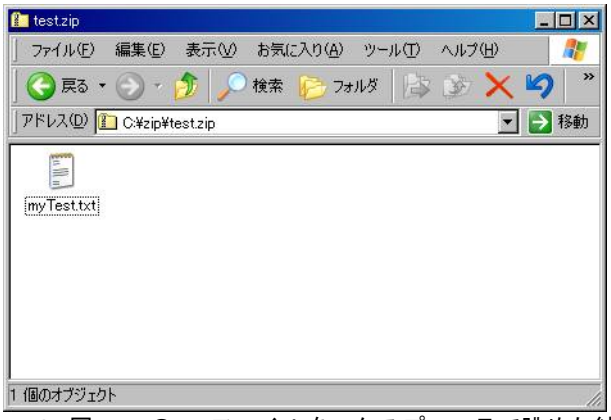

図 7.1-6：図 7.1-5のZIPファイルをエクスプローラで眺めた結果

```
Option Explicit
Dim myFileSystemObject
Dim myFileObject
Dim myFolderObject
Dim myShellApplication
Dim Hako
Dim i
Dim iObj
Dim myBin
Dim myZipPath
Dim myFile
Dim ZipFile
Dim FolderPath
ZipFile = ".¥test.zip"
FolderPath = ".¥test¥"

        REM Create Empty-ZIP File Data
Hako = Array(80,75,5,6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
For i = 0 To UBound(Hako)
 myBin = myBin + Chr(Hako(i))
Next

        REM Create Empty-ZIP File
Set myFileSystemObject = WScript.CreateObject("Scripting.FileSystemObject")
myZipPath = myFileSystemObject.GetAbsolutePathName(ZipFile)
Set myFileObject = myFileSystemObject.CreateTextFile(myZipPath,True)
myFileObject.Write myBin
myFileObject.Close
Set myFileObject = Nothing

        REM Copy into ZIP file
Set myShellApplication = WScript.CreateObject("Shell.Application")
i = 0

        REM Loop in Folder
Set myFolderObject = myFileSystemObject.GetFolder(FolderPath)
For Each iObj In myFolderObject.Files
 myFile = FolderPath & "¥" & iObj.Name
 myFile = myFileSystemObject.GetAbsolutePathName(myFile)
 WScript.Echo myFile
 myShellApplication.Namespace(myZipPath).CopyHere(myFile)
 i = i + 1
Next
Set myFolderObject = Nothing

        REM Wait for Exit
Do Until myShellApplication.Namespace(myZipPath).Items.Count = i
 WScript.Sleep 1
Loop
Set myShellApplication = Nothing
Set myFileSystemObject = Nothing
WScript.Quit
```

図 7.1-7：サンプルコード

8. Microsoft .NET Framework の場合

8.1. Microsoft .NET Framework の場合のまとめ

Microsoft .NET Framework の場合、zip ファイルの扱いには、複数の方法がある。

- Microsoft .NET Framework 2.0 上で動作する J# 2.0 のライブラリ(vjslib.dll, vjsnativ.dll) を使う
- SharpZipLib (#ziplib) (ICSharpCode.SharpZipLib.dll) というライブラリを使う
- DotNetZip (Ionic Zip Library) (Ionic.Zip.dll) というライブラリを使う

これらを使って、特定のディレクトリ内のファイルを圧縮するサンプル(図 8.1-2)を作り(図 8.1-3)、テストしてみた。
結論としては、当然といえば当然であるが、,NET Framework では、内部処理を UNICODE で行っているため、zip 圧縮する際にファイル名を ANSI コード(明示的、暗示的)に変換するような場合は、本文書のテーマである UNICODE によって、分離された文字を使ったサニタイズ回避テクニックが有効である。

検証環境

- MS-Windows XP SP3
- MS .NET Framework 2.0 SP2
- J# 2.0 SecondEdition
  - vjslib.dll：ver 2.0.50727.937
  - vjsnativ.dll：ver 2.0.50727.937
- SharpZipLib (#ziplib)
  - ICSharpCode.SharpZipLib.dll：ver 0.86.0.518
- DotNetZip (Ionic Zip Library)
  - Ionic.Zip.dll ：ver 1.9.1.8

使用したライブラリ	文字コード指定	ファイル名の「円記号」
J#2.0	-	バックスラッシュに変換
SharpZipLib - FastZip	-	バックスラッシュに変換
SharpZipLib - ZipOutputStream	Shift-JIS	バックスラッシュに変換
SharpZipLib - ZipOutputStream (ZipNameTRansform)	Shift-JIS	バックスラッシュに変換
SharpZipLib - ZipOutputStream	UTF-8	「円記号」のまま
SharpZipLib - ZipOutputStream (ZipNameTRansform)	UTF-8	「円記号」のまま
SharpZipLib - ZipFile	-	バックスラッシュに変換
SharpZipLib - ZipFile (ZipNameTRansform)	-	バックスラッシュに変換
DotNetZip - ZipOutputStream	Shift-JIS	バックスラッシュに変換
DotNetZip - ZipOutPutStream	UTF-8	「円記号」のまま
DotNetZip - AddFile	Shift-JIS	バックスラッシュに変換
DotNetZip - AddFile	UTF-8	「円記号」のまま
DotNetZip - AddDirectory	Shift-JIS	バックスラッシュに変換
DotNetZip - AddDirectory	UTF-8	「円記号」のまま

図 8.1-1 : MS .NET Framework **の場合の実験結果表**

いくつかの方法では、文字コードを指定することがない。この場合、非 UNICODE 系が指定されている場面が多いため、本文書のテーマである UNICODE によって、分離された文字を使ったサニタイズ回避テクニックが有効となる可能性が高く、プログラミングする上で、注意が必要である。

以下が、ファイル名の文字コードを指定することなく、Zip 圧縮が可能であり、かつ暗黙的に文字コードが ANSI コードである方法である。

- J#2.0
- SharpZipLib - FastZip
- SharpZipLib - ZipFile

ちなみに、「SharpZipLib」の「ZipNameTransform()」というメソッドは、「..(ピリオド二個)」を一個にしてしまうため、「...(ピリオド 3 個)」+「円記号」とすることで、「..\」という上位ディレクトリを示すパスを含ませることができるようだ。

また、「DotNetZip」で、文字コードを指定する際に「AlternateEncodingUsage()」メソッドを指定しないと、デフォルトの文字コード「IBM437」で変換されてしまい、文字化けが発生してしまうようだ。

```
using System;
using System.Text;
using System.IO;

public class ZipCompressTest_NET{
 public static void Main(string[] args){
  int i;
  int iMax;
  Int32 mode;
  String CompressedFileName;
  FileStream myWriteFileStream = null;
        // .NET Zip Library #ziplib (SharpZipLib)(ICSharpCode.SharpZipLib.dll)
  ICSharpCode.SharpZipLib.Zip.FastZip myFastZip = null;
  ICSharpCode.SharpZipLib.Zip.ZipOutputStream myZipOutputStream = null;
  ICSharpCode.SharpZipLib.Zip.ZipFile mySharpZipLibZipFile = null;
  ICSharpCode.SharpZipLib.Zip.ZipNameTransform myZipNameTransform = null;
  ICSharpCode.SharpZipLib.Zip.ZipEntry myZipEntry = null;
        // DotNetZip (Ionic.Zip.dll)
  Ionic.Zip.ZipFile myIonicZipFile = null;
  Ionic.Zip.ZipOutputStream myIonicOutputStream = null;
        // J# 2.0 (vjslib.dll)
  java.io.FileOutputStream myJSFileOutputStream = null;
  java.util.zip.ZipOutputStream myJSZipOutputStream = null;

        // ヘルプを表示
  usage();
        // 引数を取得
  if(2 < args.Length){
   try{
    mode = Int32.Parse(args[0]);
    String zipFileName = args[1];
    String ZipDirectory = args[2];
    Console.WriteLine("ZipFile : " + zipFileName);
```

```
    Console.WriteLine("Directory : " + ZipDirectory);
    if(System.IO.Directory.Exists(ZipDirectory) == true){
        // ////////////////////////////////////
        // ここはディレクトリ指定で圧縮
     if(mode == 0 || mode == 17 || mode == 18){
      if(mode == 0){
       myFastZip = new ICSharpCode.SharpZipLib.Zip.FastZip();
       myFastZip.CreateZip(zipFileName,ZipDirectory,true,null,null);
      }else if(mode == 17 || mode == 18){
       myIonicZipFile = new Ionic.Zip.ZipFile();
       if(mode == 17){
        // IBM437 でエンコードできないファイル名は SJIS でエンコード
        myIonicZipFile.AlternateEncoding = Encoding.GetEncoding("shift_jis");
       }else{
        // IBM437 でエンコードできないファイル名は UTF-8 でエンコード
//        myIonicZipFile.UseUnicodeAsNecessary = true;
        myIonicZipFile.AlternateEncoding = Encoding.GetEncoding("utf-8");
       }
       myIonicZipFile.AlternateEncodingUsage = Ionic.Zip.ZipOption.Always;
       myIonicZipFile.CompressionLevel = Ionic.Zlib.CompressionLevel.BestCompression;
       myIonicZipFile.AddDirectory(ZipDirectory,"");
       myIonicZipFile.Save(zipFileName);
      }
     }else{
        // ////////////////////////////////////
        // ここはファイル指定なので、ファイル一覧を取得して FOR で回す
      String[] strHako = System.IO.Directory.GetFiles(ZipDirectory);
      iMax = strHako.Length;
        // 回す前の個別処理
      if(0 < mode && mode < 10){
       myZipNameTransform = new ICSharpCode.SharpZipLib.Zip.ZipNameTransform(ZipDirectory);
      }
      if(0 < mode && mode < 5){
       myWriteFileStream = new FileStream(zipFileName,FileMode.Create,FileAccess.Write);
       myZipOutputStream = new ICSharpCode.SharpZipLib.Zip.ZipOutputStream(myWriteFileStream);
       myZipOutputStream.SetLevel(9);
      }else if(mode == 5 || mode == 6){
       mySharpZipLibZipFile = ICSharpCode.SharpZipLib.Zip.ZipFile.Create(zipFileName);
       mySharpZipLibZipFile.BeginUpdate();
      }else if(mode == 10 || mode == 11){
       myWriteFileStream = new FileStream(zipFileName,FileMode.Create,FileAccess.Write);
       myIonicOutputStream = new Ionic.Zip.ZipOutputStream(myWriteFileStream);
       if(mode == 10){
        // IBM437 でエンコードできないファイル名は SJIS でエンコード
        myIonicOutputStream.AlternateEncoding = Encoding.GetEncoding("shift_jis");
       }else{
        // IBM437 でエンコードできないファイル名は UTF-8 でエンコード
//        myIonicOutputStream.UseUnicodeAsNecessary = true;
        myIonicOutputStream.AlternateEncoding = Encoding.GetEncoding("utf-8");
       }
       myIonicOutputStream.AlternateEncodingUsage = Ionic.Zip.ZipOption.Always;
      }else if(mode == 13 || mode == 14){
       myIonicZipFile = new Ionic.Zip.ZipFile();
       if(mode == 13){
        // IBM437 でエンコードできないファイル名は SJIS でエンコード
        myIonicZipFile.AlternateEncoding = Encoding.GetEncoding("shift_jis");
```

```
        }else{
         // IBM437 でエンコードできないファイル名は UTF-8 でエンコード
//         myIonicZipFile.UseUnicodeAsNecessary = true;
         myIonicZipFile.AlternateEncoding = Encoding.GetEncoding("utf-8");
        }
        myIonicZipFile.AlternateEncodingUsage = Ionic.Zip.ZipOption.Always;

        myIonicZipFile.CompressionLevel = Ionic.Zlib.CompressionLevel.BestCompression;
       }else if(mode == 20){
        myJSFileOutputStream = new java.io.FileOutputStream(zipFileName);
        myJSZipOutputStream = new java.util.zip.ZipOutputStream(myJSFileOutputStream);
       }
       for(i=0;i<iMax;i++){
         // 回している最中の共通処理
        CompressedFileName = GetFileName(strHako[i]);
        if(0 < mode && mode < 10){
         if(mode == 2 || mode == 4 || mode == 6){
          CompressedFileName = myZipNameTransform.TransformFile(CompressedFileName);
         }
        }
        Console.Write("FileName : " + CompressedFileName + "¥r¥nFileName(Hex) : ");
        DisplayHEX(CompressedFileName);
         // 回している最中の個別処理
        if(0 < mode && mode < 5){
         myZipEntry = new ICSharpCode.SharpZipLib.Zip.ZipEntry(CompressedFileName);
         if(mode == 1 || mode == 2){
          myZipEntry.IsUnicodeText = false;
         }else{
          myZipEntry.IsUnicodeText = true;
         }
         myZipOutputStream.PutNextEntry(myZipEntry);
         byte[] myByte = ReadFile(strHako[i]);
         myZipOutputStream.Write(myByte,0,myByte.Length);
        }else if(mode == 5 || mode == 6){
         mySharpZipLibZipFile.Add(strHako[i],CompressedFileName);
        }else if(mode == 10 || mode == 11){
         myIonicOutputStream.PutNextEntry(CompressedFileName);
         byte[] myByte = ReadFile(strHako[i]);
         myIonicOutputStream.Write(myByte,0,myByte.Length);
        }else if(mode == 13 || mode == 14){
         myIonicZipFile.AddFile(strHako[i]);
        }else if(mode == 20){
         java.util.zip.ZipEntry myZipEntryJ = new java.util.zip.ZipEntry(CompressedFileName);
         myZipEntryJ.setMethod(java.util.zip.ZipEntry.DEFLATED);
         myJSZipOutputStream.putNextEntry(myZipEntryJ);
         java.io.FileInputStream myJSFileInputStream = new java.io.FileInputStream(strHako[i]);
         sbyte[] mysByte = new sbyte[8192];
         int mysByteLen;
         while((mysByteLen = myJSFileInputStream.read(mysByte,0,mysByte.Length)) > 0){
          myJSZipOutputStream.write(mysByte,0,mysByteLen);
         }
         myJSFileInputStream.close();
         myJSZipOutputStream.closeEntry();
        }
       }
         // 最後の個別処理
```

```
      if(0 < mode && mode < 5){
       myZipOutputStream.Finish();
       myZipOutputStream.Close();
       myWriteFileStream.Close();
      }else if(mode == 5 || mode == 6){
       mySharpZipLibZipFile.CommitUpdate();
       mySharpZipLibZipFile.Close();
      }else if(mode == 10 || mode == 11){
       myIonicOutputStream.Close();
       myWriteFileStream.Close();
      }else if(mode == 13 || mode == 14){
       myIonicZipFile.Save(zipFileName);
      }else if(mode == 20){
       myJSZipOutputStream.close();
       myJSFileOutputStream.close();
      }
     }
    }
   }catch(Exception e){
    Console.WriteLine("error : " + e.ToString());
   }
  }
  Console.WriteLine("done!");
}

private static byte[] ReadFile(String iFileName){
 FileInfo myFileInfo = new FileInfo(iFileName);
 long FileLen = myFileInfo.Length;
 byte[] myByte = new byte[FileLen];
 FileStream myReadFileStream = new FileStream(iFileName,FileMode.Open,FileAccess.Read);
 FileLen = myReadFileStream.Read(myByte,0,myByte.Length);
 myReadFileStream.Close();
 return myByte;
}

private static String GetFileName(String iStr){
 Char[] cHako = new Char[1];
 cHako[0] = '¥¥';
 String[] Hako = iStr.Split(cHako);
 return Hako[Hako.Length-1];
}

private static void DisplayHEX(String iStr){
 Encoding myEncoding = Encoding.GetEncoding("utf-16");
 byte[] myByte = myEncoding.GetBytes(iStr);
 int iMax = myByte.Length;
 for(int i=0;i<iMax;i++){
  if(myByte[i] < 16){
   Console.Write("0");
  }
  Console.Write(myByte[i].ToString("x"));
  if(i < iMax-1){
   Console.Write(" ");
  }
 }
 Console.Write("¥r¥n");
```

```
 }

 private static void usage(){
  Console.WriteLine("Usage:");
  Console.WriteLine("      ZipCompressTest_NET.exe <<mode>> <<ZipFile>>
<<CompressedDirectory>>");
  Console.WriteLine(" mode:");
  Console.WriteLine("  .NET Zip Library #ziplib (SharpZipLib)(ICSharpCode.SharpZipLib.dll)");
  Console.WriteLine("    0 : FastZip");
  Console.WriteLine("    1 : ZipOutputStream (Shift_JIS)");
  Console.WriteLine("    2 : ZipOutputStream (Shift_JIS) (use ZipNameTransform)");
  Console.WriteLine("    3 : ZipOutputStream (UTF-8)");
  Console.WriteLine("    4 : ZipOutputStream (UTF-8) (use ZipNameTransform)");
  Console.WriteLine("    5 : ZipFile");
  Console.WriteLine("    6 : ZipFile (use ZipNameTransform)");
  Console.WriteLine("  DotNetZip (Ionic.Zip.dll) ZipFile");
  Console.WriteLine("   10 : ZipOutputStream (ShiftJIS)");
  Console.WriteLine("   11 : ZipOutputStream (UTF-8)");
  Console.WriteLine("   13 : AddFile (ShiftJIS)");
  Console.WriteLine("   14 : AddFile (UTF-8)");
  Console.WriteLine("   17 : AddDirectory (ShiftJIS)");
  Console.WriteLine("   18 : AddDirectory (UTF-8)");
  Console.WriteLine("  J# 2.0 (java.util.zip.ZipOutputStream)(vjslib.dll)");
  Console.WriteLine("   20 : ZipOutputStream");
 }
}
```

図 8.1-2：サンプルコード(ZipCompressTest_NET.cs)

```
SET FILENAME=ZipCompressTest_NET
IF EXIST %FILENAME%.exe DEL %FILENAME%.exe
CSC.EXE %FILENAME%.cs /reference:ICSharpCode.SharpZipLib.dll /reference:Ionic.Zip.dll
/reference:vjslib.dll
SET FILENAME=
```

図 8.1-3：図 8.1-2のmakeファイル

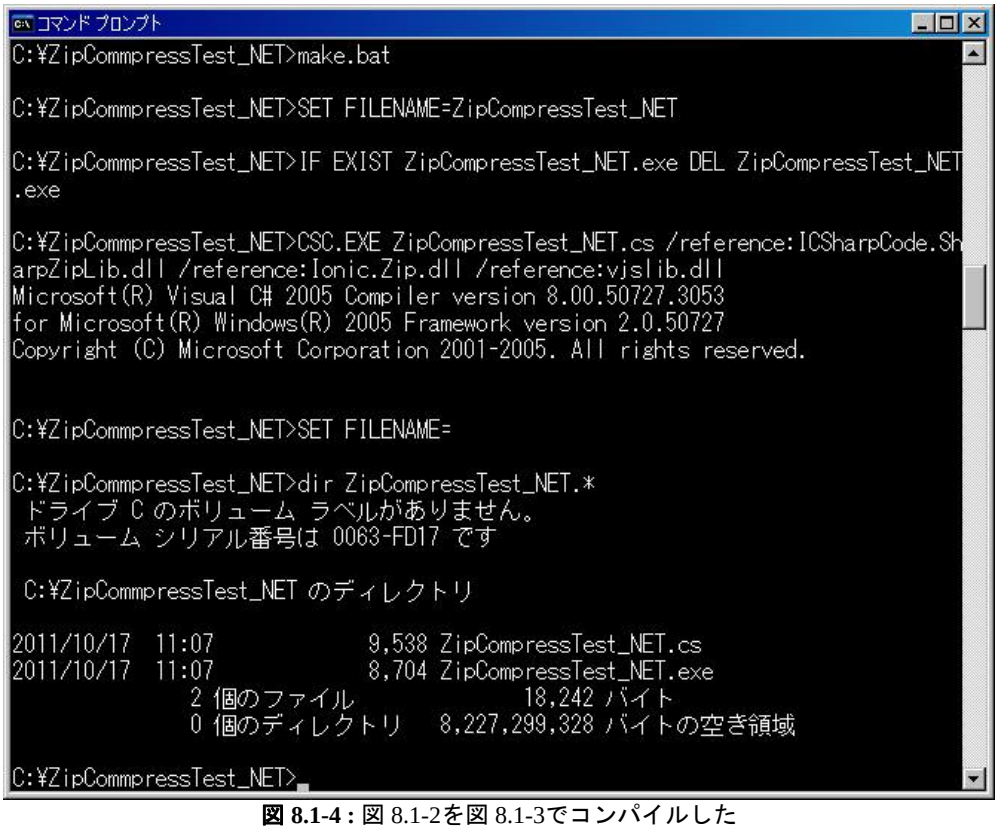

図 8.1-4 : 図 8.1-2を図 8.1-3でコンパイルした

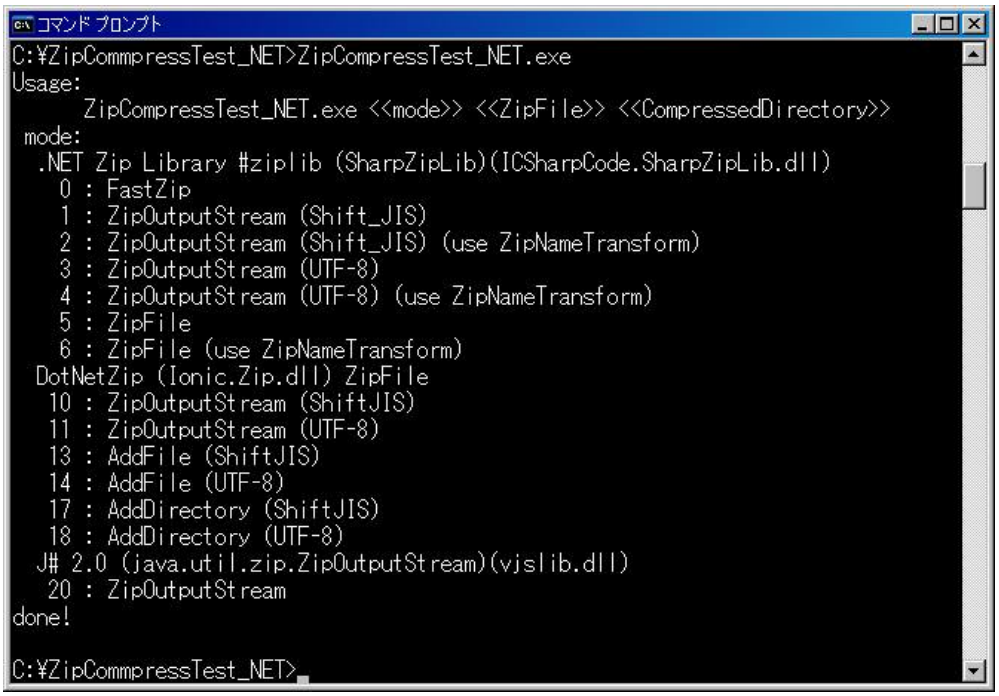

図 8.1-5 : 図 8.1-2のusage

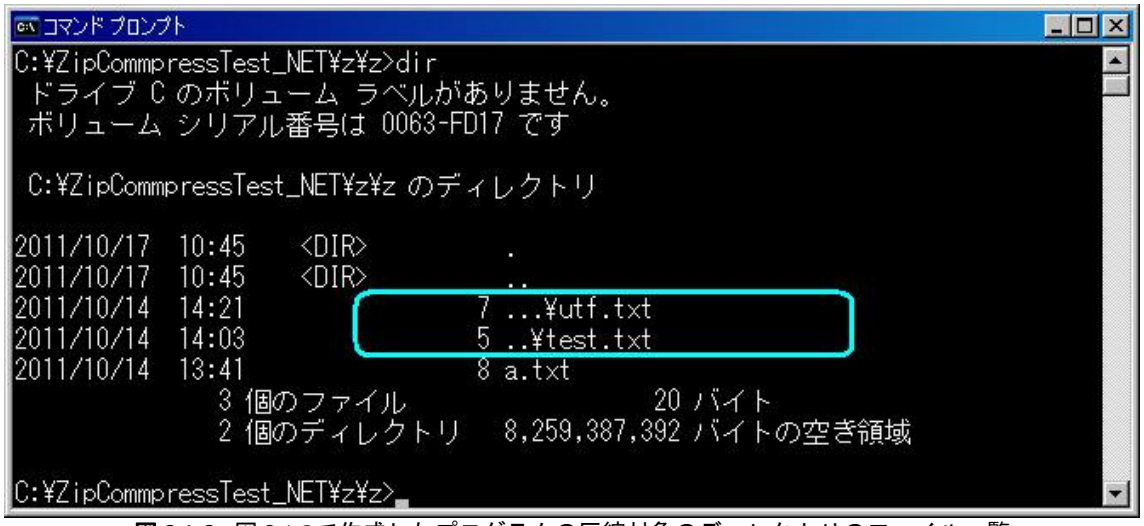

図 8.1-6 : 図 8.1-2で作成したプログラムの圧縮対象のディレクトリのファイル一覧

以降、基本的にはこのファイル(「a.txt」と「..(円記号)test.txt」と

「…(円記号)utf.txt」)を圧縮する

8.2. J#2.0 (vjslib.dll, vjsnativ.dll) の場合

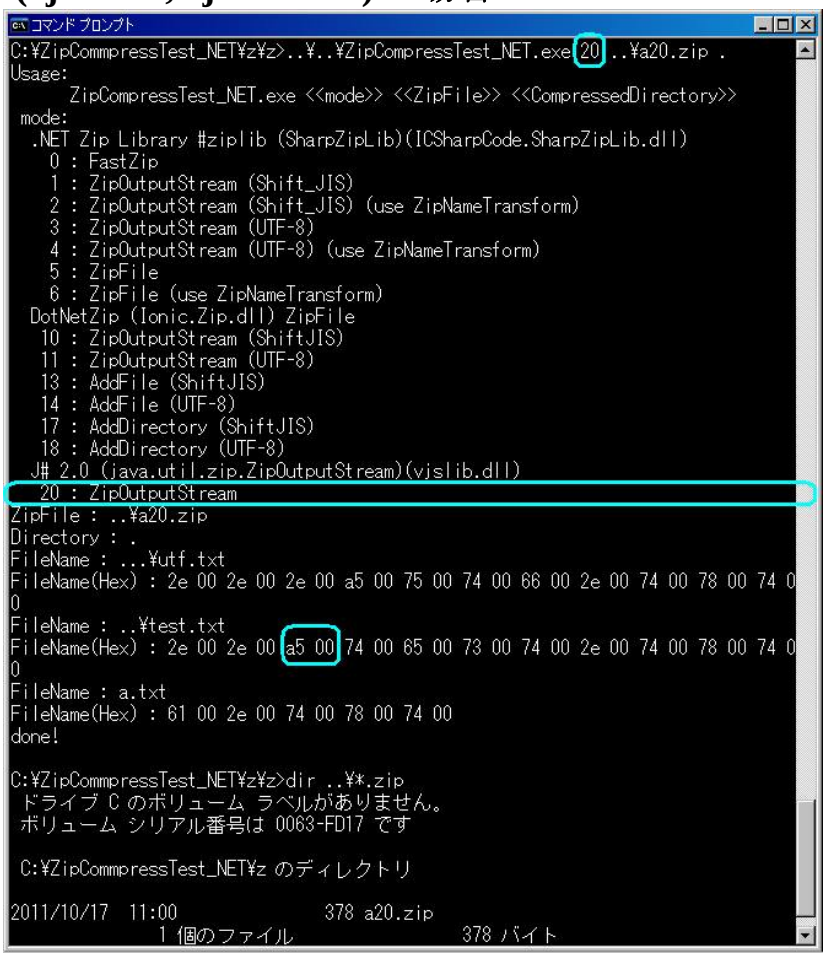

図 8.2-1 : J#2.0 の ZipOutputStream クラスを使って圧縮する。

ファイル名の「円記号」はきちんと「a5 00」となっている

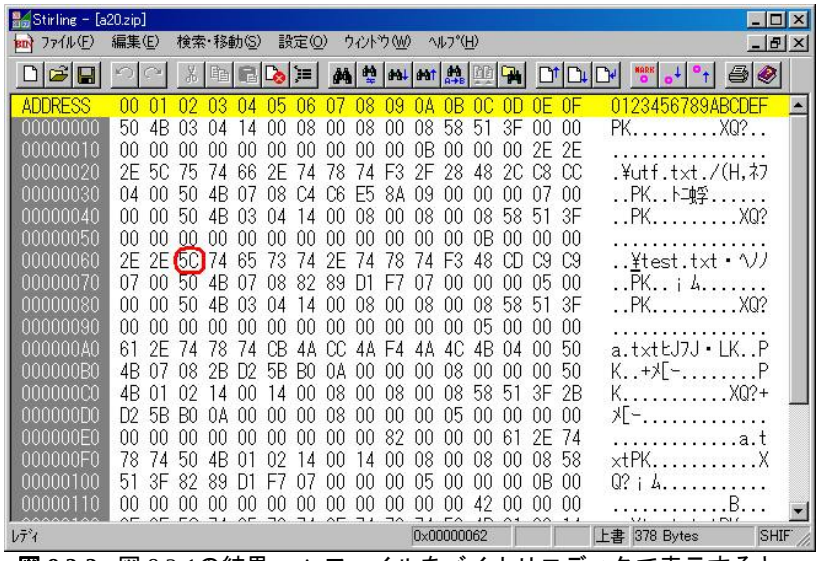

図 8.2-2 : 図 8.2-1の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

8.3. SharpZipLib (#ziplib) (ICSharpCode.SharpZipLib.dll) の場合

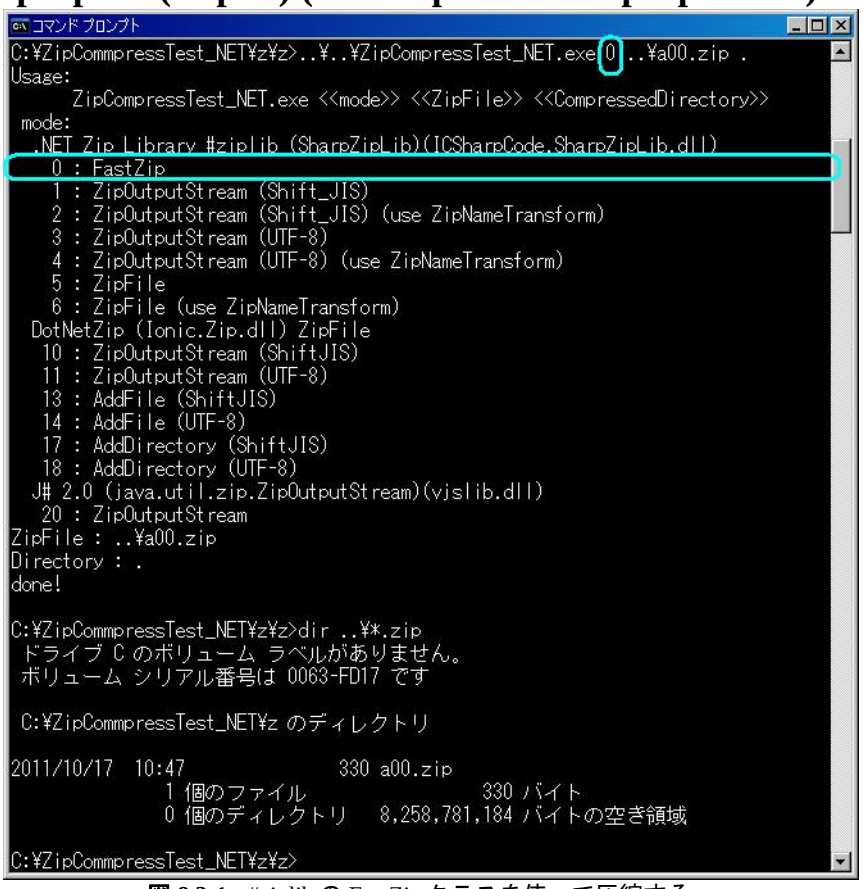

図 8.3-1 : #ziplib の FastZip クラスを使って圧縮する。

ファイル名の 16 進表示はできないが、

ファイル名の「円記号」はきちんと「a5 00」となっているだろう

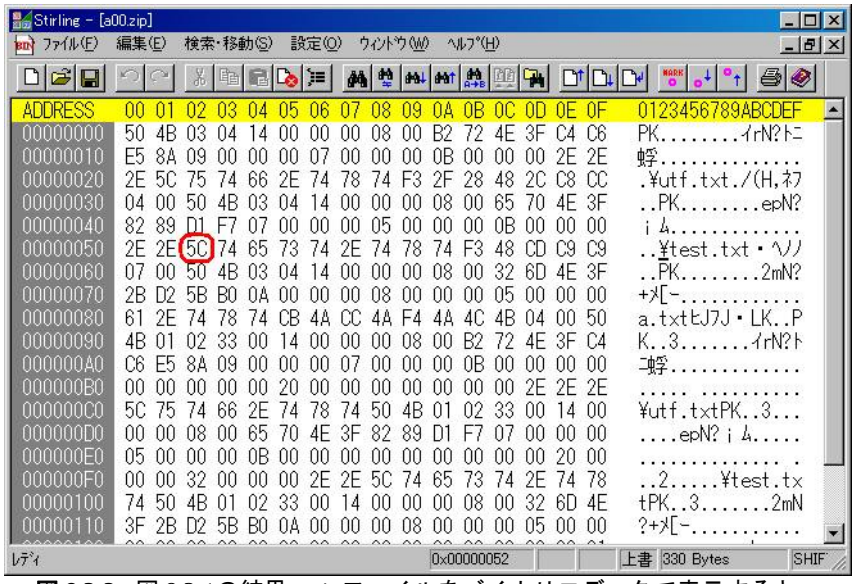

図 8.3-2 : 図 8.3-1の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

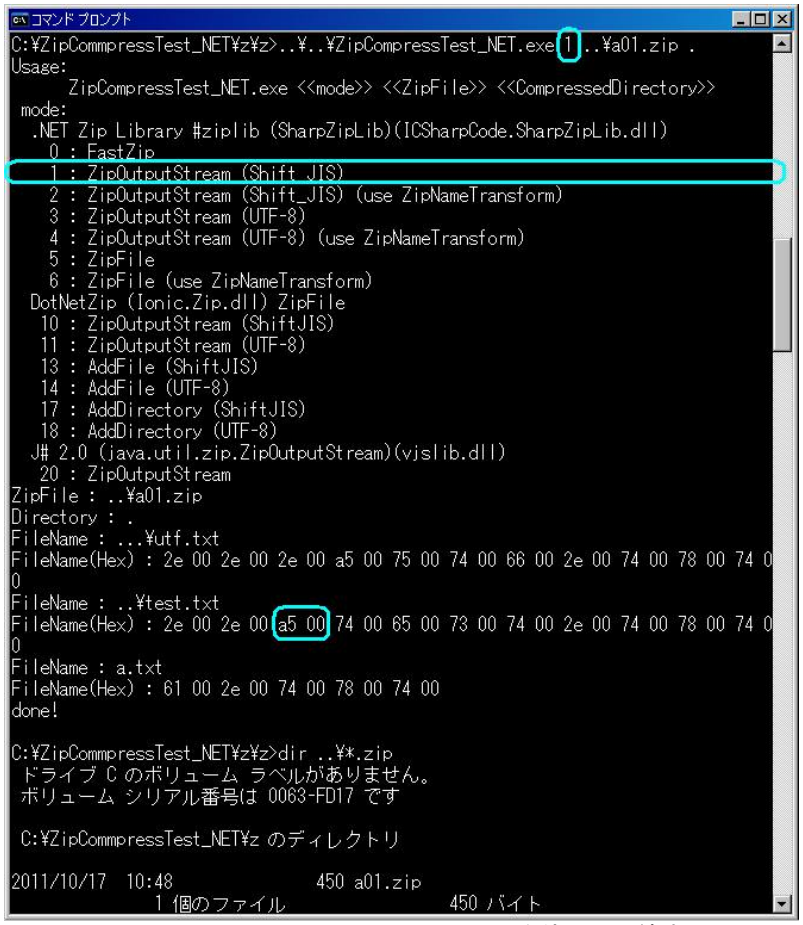

図 8.3-3 : #ziplib の ZipOutputStream クラスを使って圧縮する。

(文字コードを非 UNICODE に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

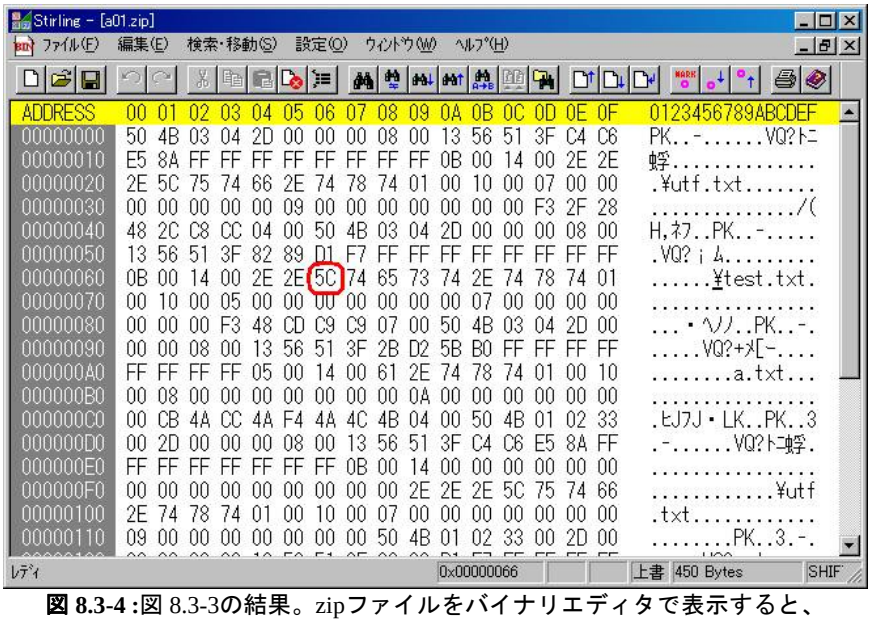

図 8.3-4 :図 8.3-3の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

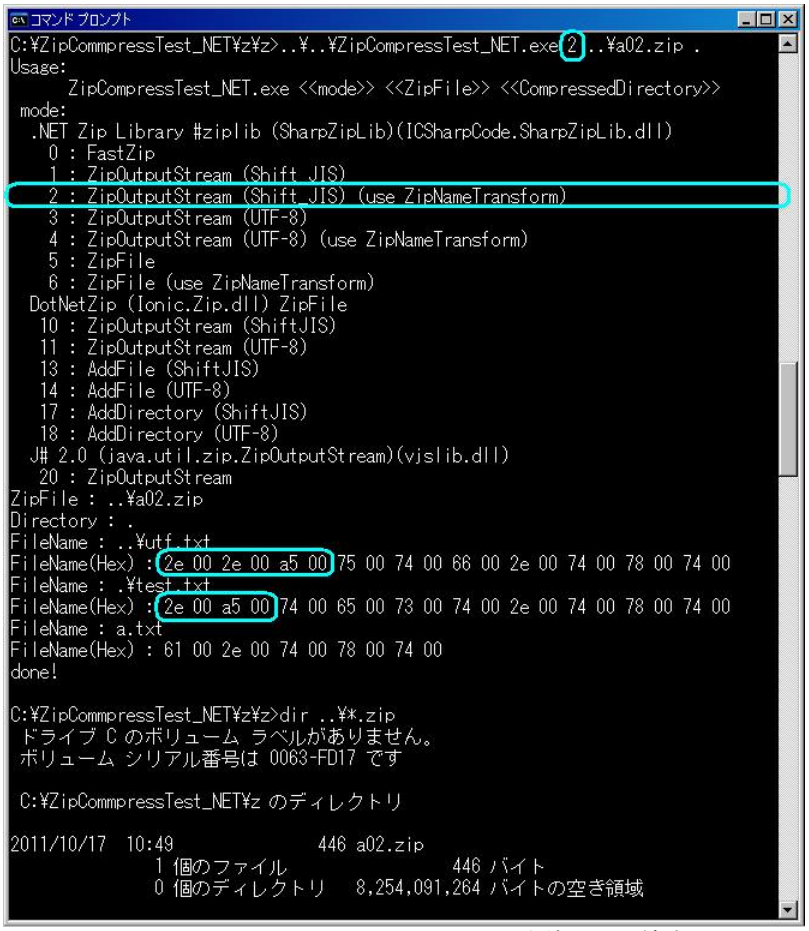

図 8.3-5 : #ziplib の ZipOutputStream クラスを使って圧縮する。

(文字コードを非 UNICODE に指定し ZipNameTransform クラスを使う)

ファイル名の「円記号」はきちんと「a5 00」となっている

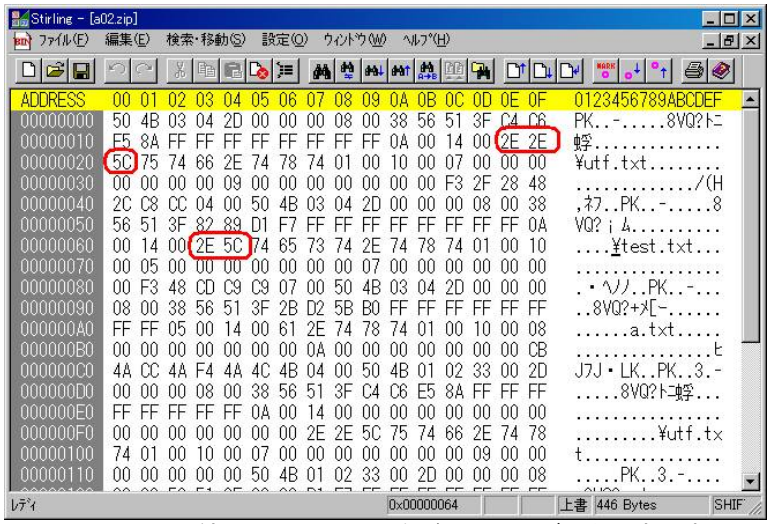

図 8.3-6 : 図 8.3-5の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっているが、

ピリオドが一つ消去されている。つまり、

ピリオド 3 つの「…(円記号)utf.txt」が「..\utf.txt」になった

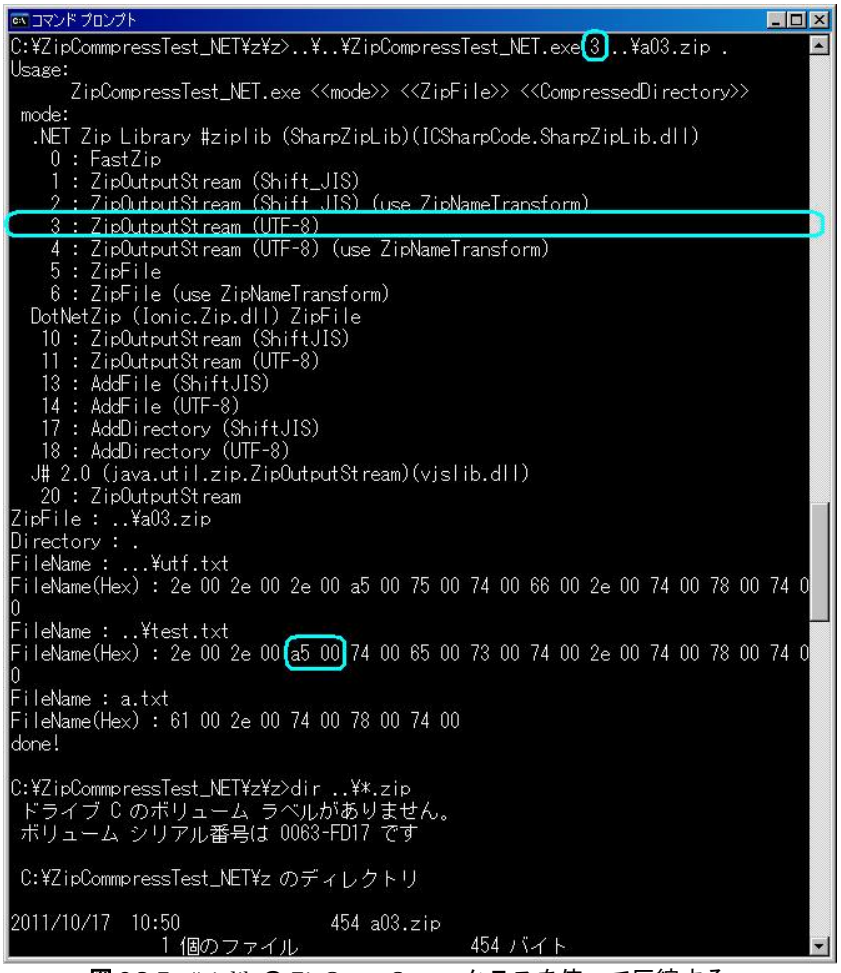

図 8.3-7 : #ziplib の ZipOutputStream クラスを使って圧縮する。

(文字コードを UNICODE に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

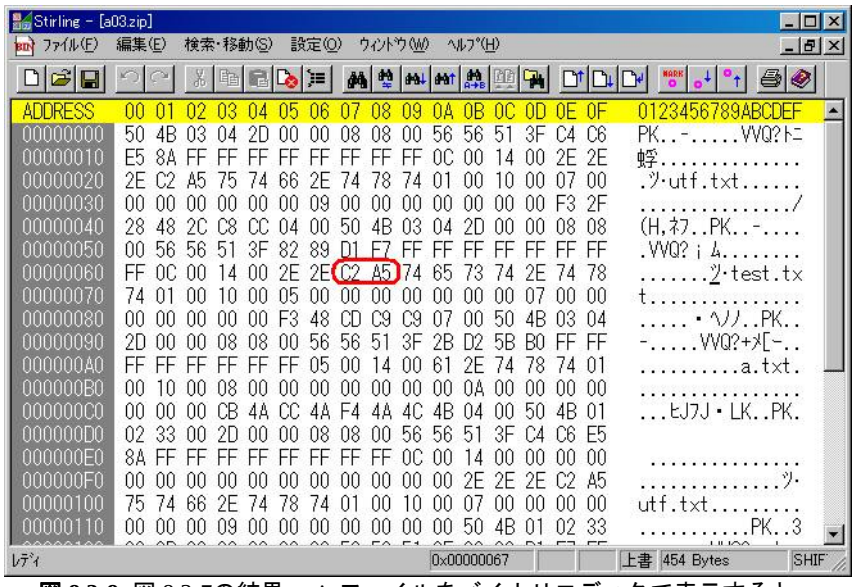

図 8.3-8 :図 8.3-7の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」はそのまま「円記号(UTF-8 の 0xC2 0xA5)」のままだった

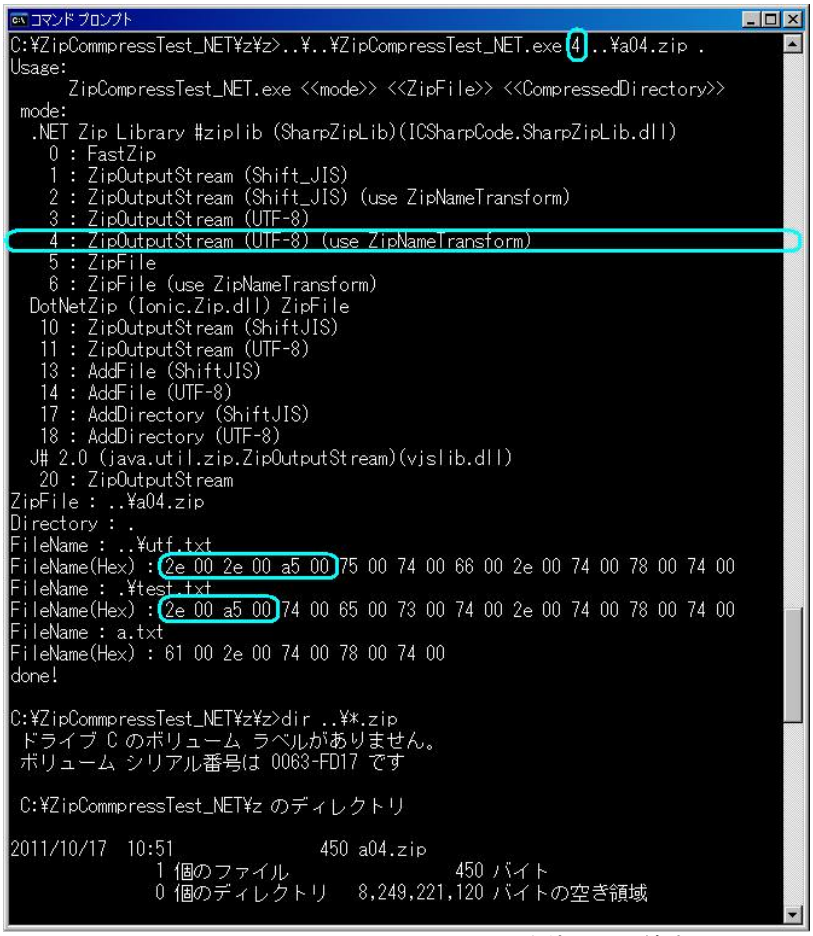

図 8.3-9 : #ziplib の ZipOutputStream クラスを使って圧縮する。

(文字コードを UNICODE に指定し ZipNameTransform クラスを使う)

ファイル名の「円記号」はきちんと「a5 00」となっている

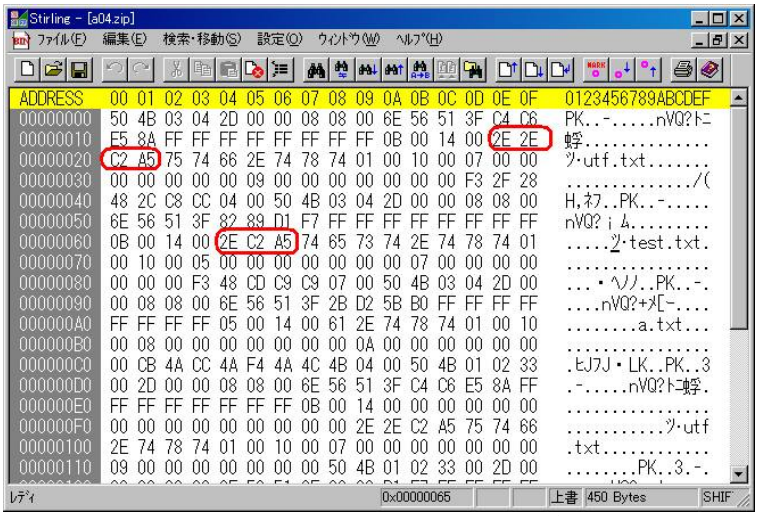

図 8.3-10 : 図 8.3-9の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」はそのまま「円記号(UTF-8 の 0xC2 0xA5)」だが、

ピリオドが一つ消去されている。つまり、

ピリオド 3 つの「…(円記号)utf.txt」が「..(円記号)utf.txt」になった

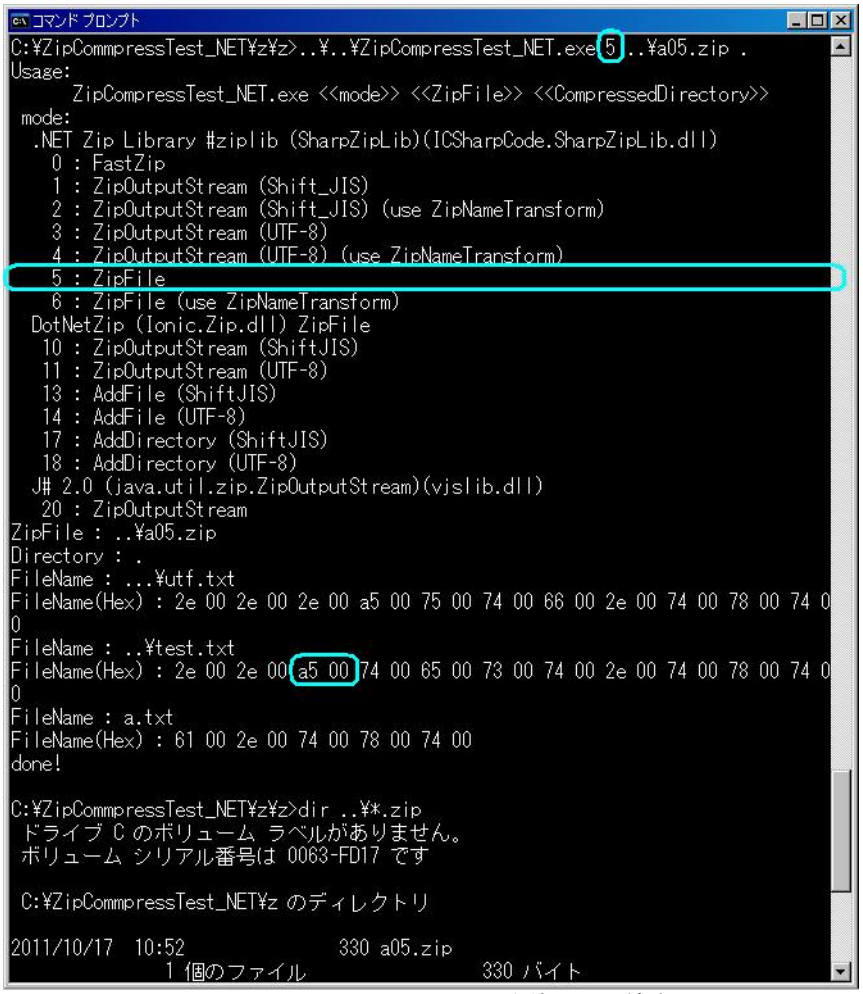

図 8.3-11 : #ziplib の ZipFile クラスを使って圧縮する。

ファイル名の「円記号」はきちんと「a5 00」となっている

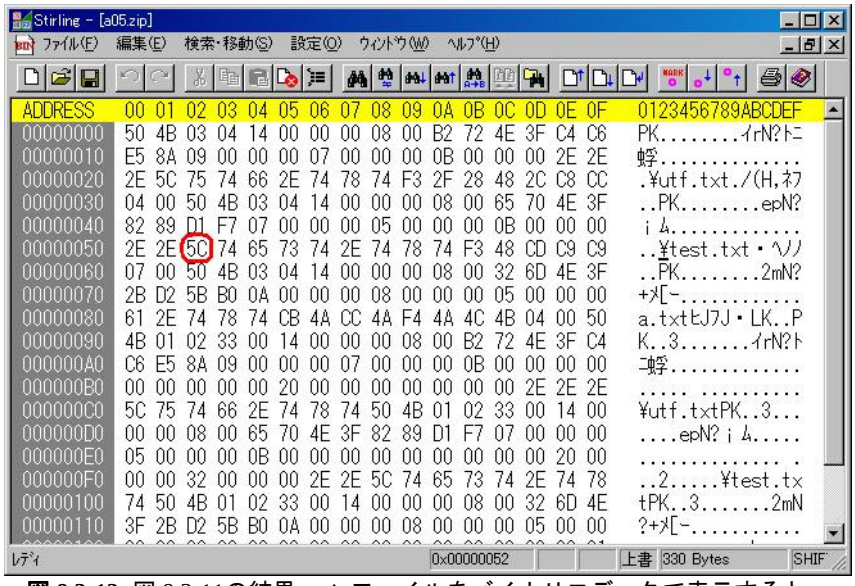

図 8.3-12 :図 8.3-11の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

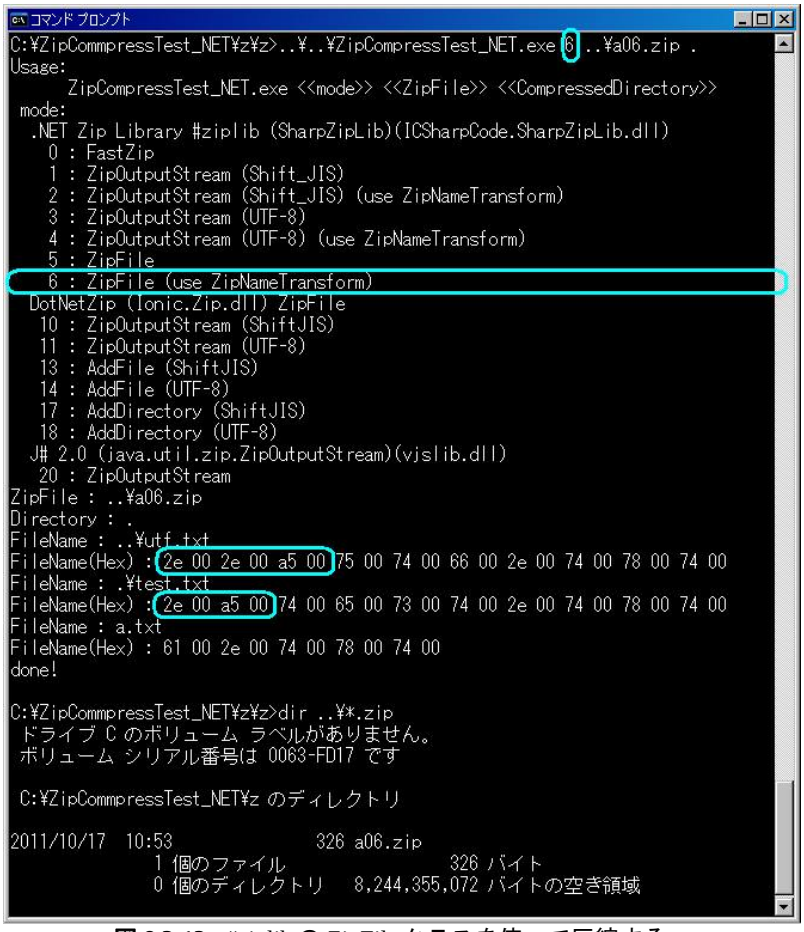

図 8.3-13 : #ziplib の ZipFile クラスを使って圧縮する。

(ZipNameTransform クラスを使う)

ファイル名の「円記号」はきちんと「a5 00」となっている

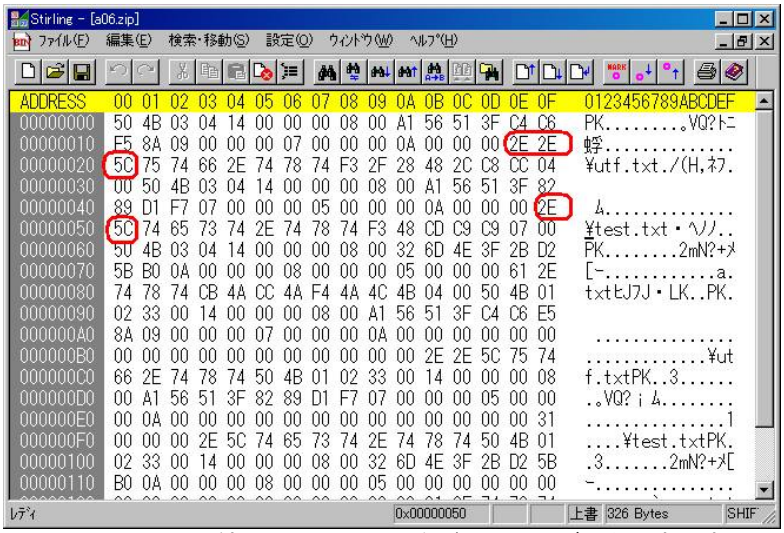

図 8.3-14 : 図 8.3-13の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっているが、

ピリオドが一つ消去されている。つまり、

ピリオド 3 つの「…(円記号)utf.txt」が「..\utf.txt」になった

8.4. DotNetZip (Ionic Zip Library) (Ionic.Zip.dll) の場合

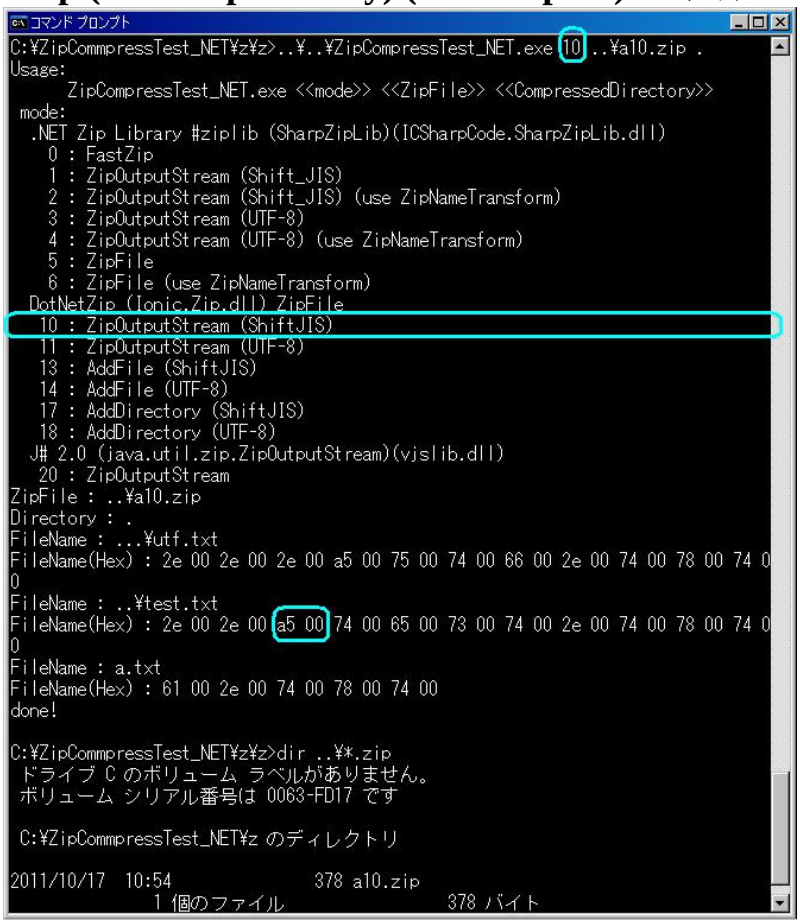

図 8.4-1 : DotNetZip の ZipOutputStream クラスを使って圧縮する。

(文字コードを Shift-JIS に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

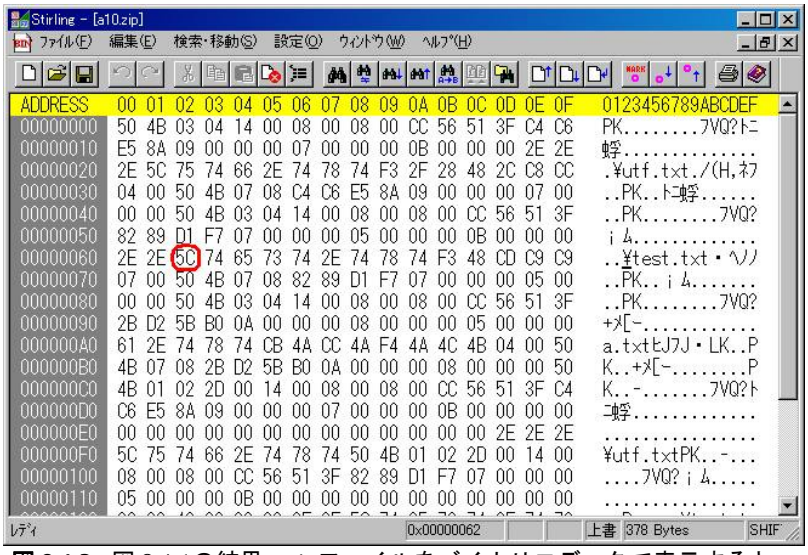

図 8.4-2 : 図 8.4-1の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

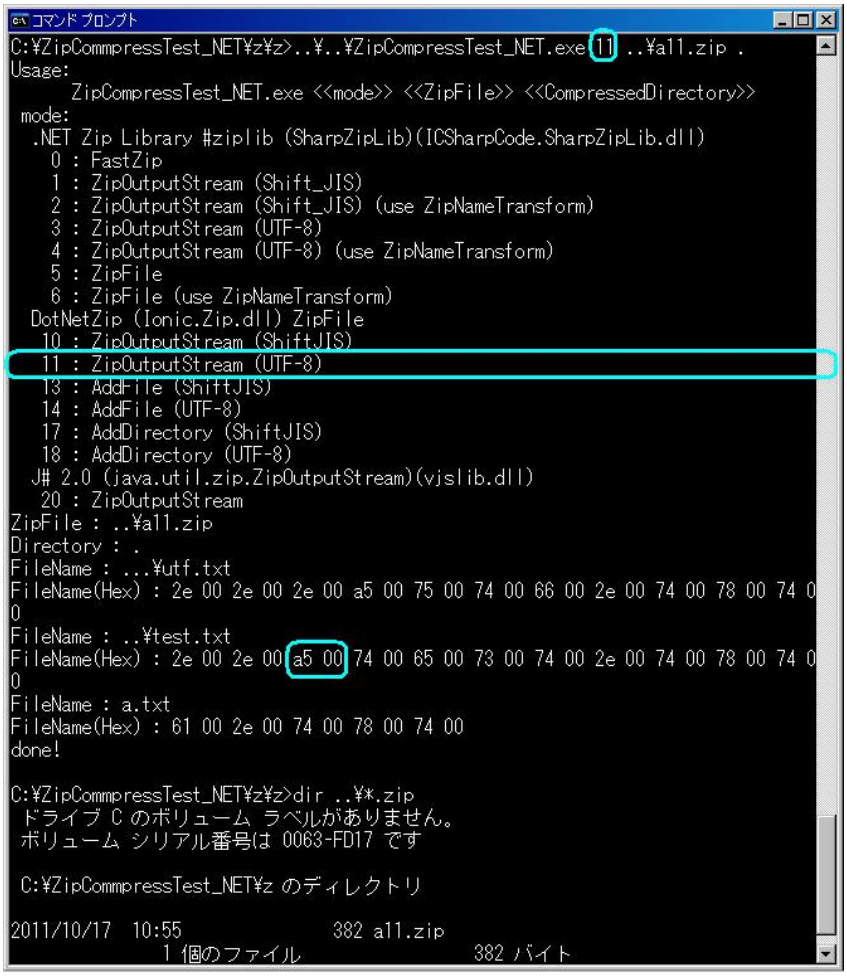

図 8.4-3 : DotNetZip の ZipOutputStream クラスを使って圧縮する。

(文字コードを UTF-8 に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

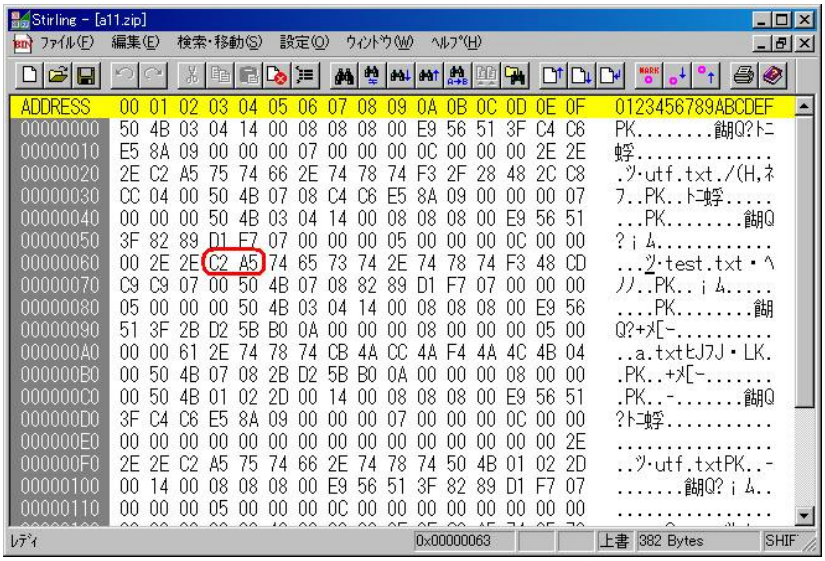

図 8.4-4 : 図 8.4-3の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」はそのまま「円記号(UTF-8 の 0xC2 0xA5)」のままだった

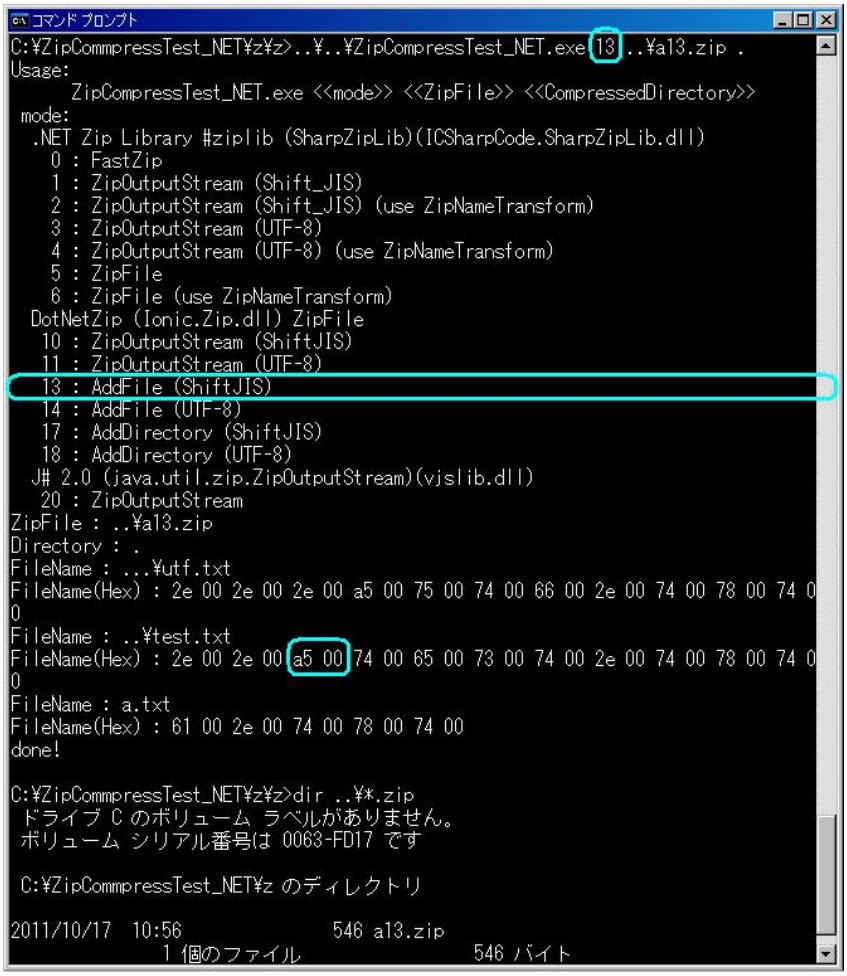

図 8.4-5 : DotNetZip の ZipFile クラスの AddFile()メソッドを使って圧縮する。

(文字コードを Shift-JIS に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

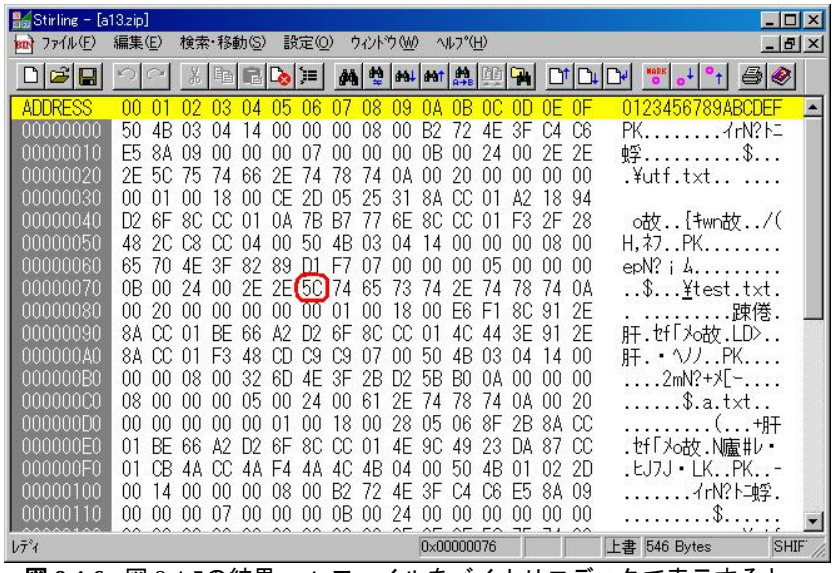

図 8.4-6 : 図 8.4-5の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

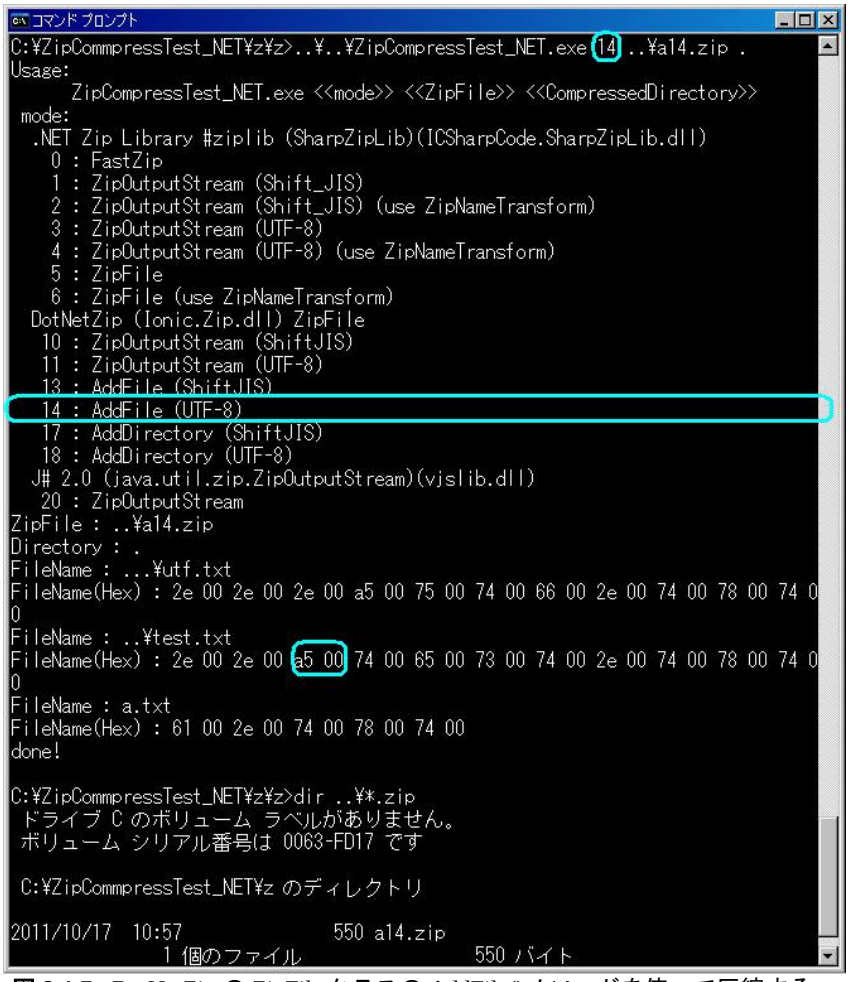

図 8.4-7 : DotNetZip の ZipFile クラスの AddFile()メソッドを使って圧縮する。

(文字コードを UTF-8 に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

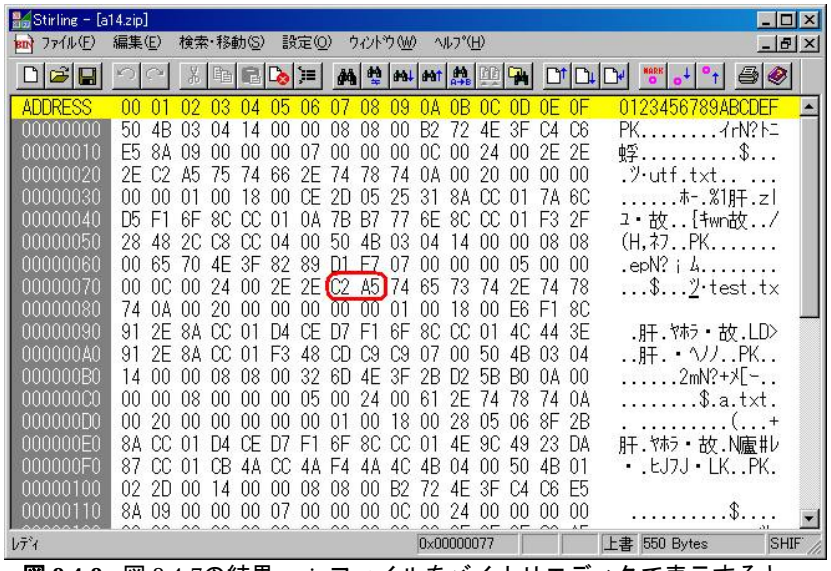

図 8.4-8 : 図 8.4-7の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」はそのまま「円記号(UTF-8 の 0xC2 0xA5)」のままだった

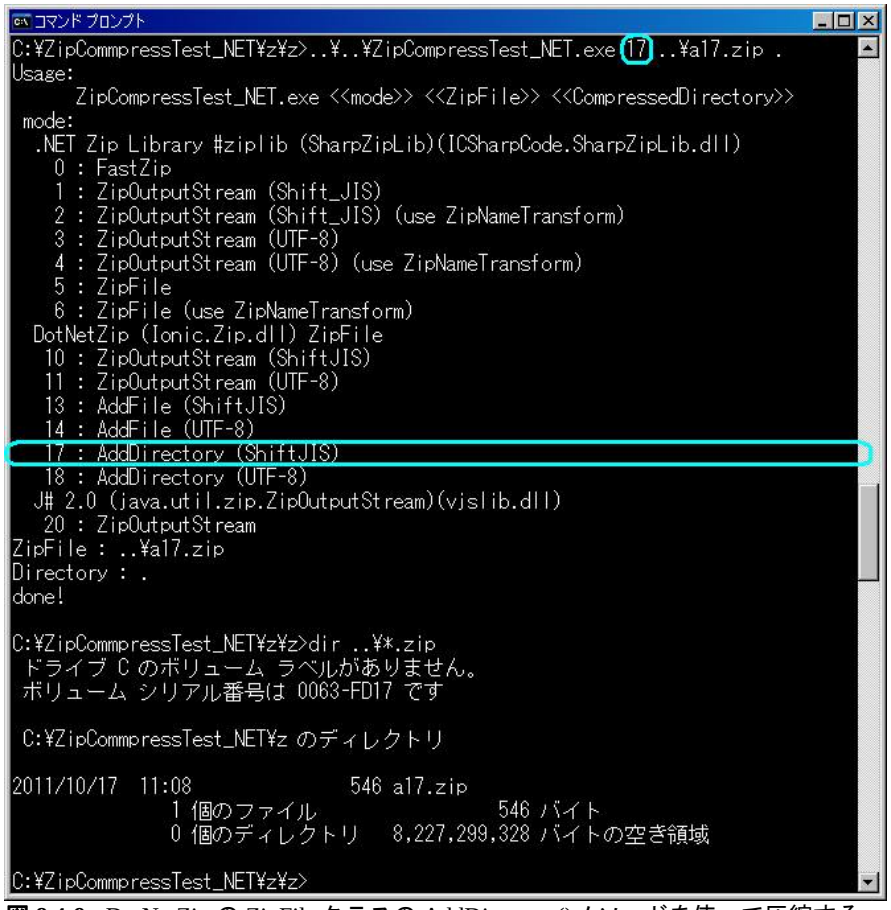

図 8.4-9 : DotNetZip の ZipFile クラスの AddDirectory()メソッドを使って圧縮する。

(文字コードを Shift-JIS に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

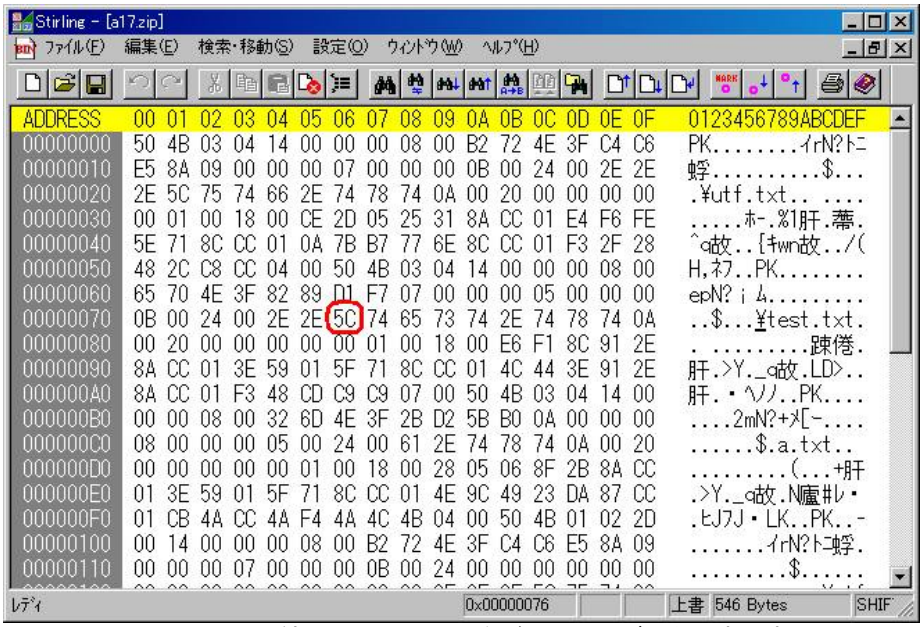

図 8.4-10 : 図 8.4-9の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」が「バックスラッシュ(0x5c)」に変換されてしまっている

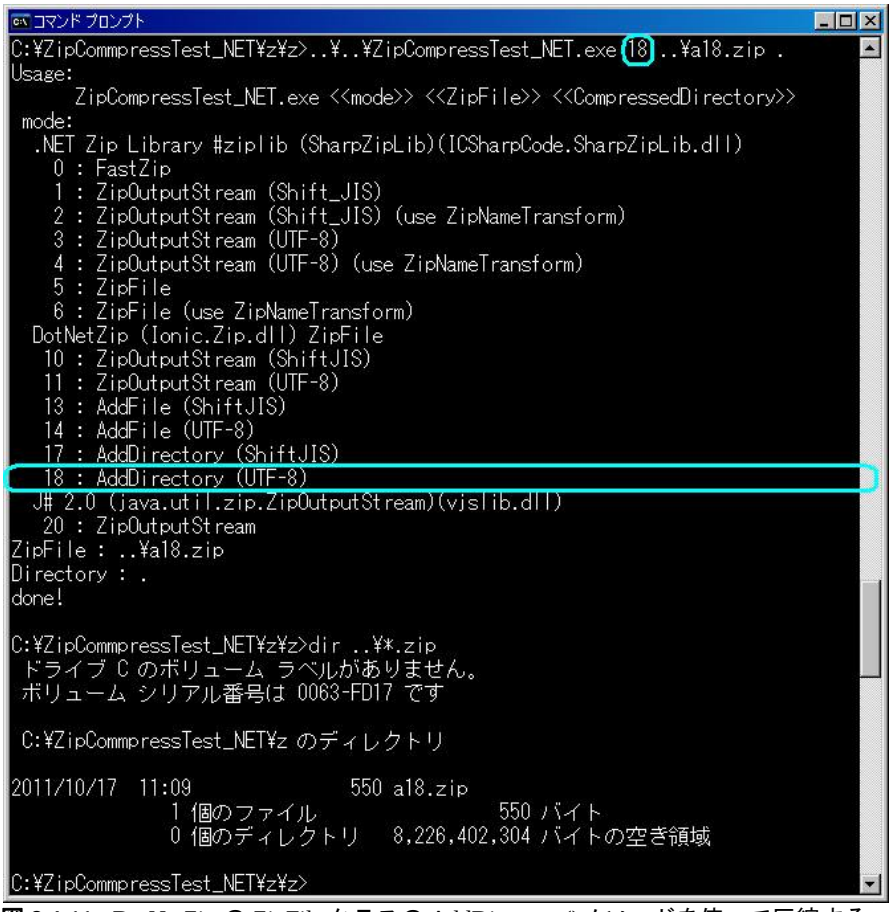

図 8.4-11 : DotNetZip の ZipFile クラスの AddDirectory()メソッドを使って圧縮する。

(文字コードを UTF-8 に指定した)

ファイル名の「円記号」はきちんと「a5 00」となっている

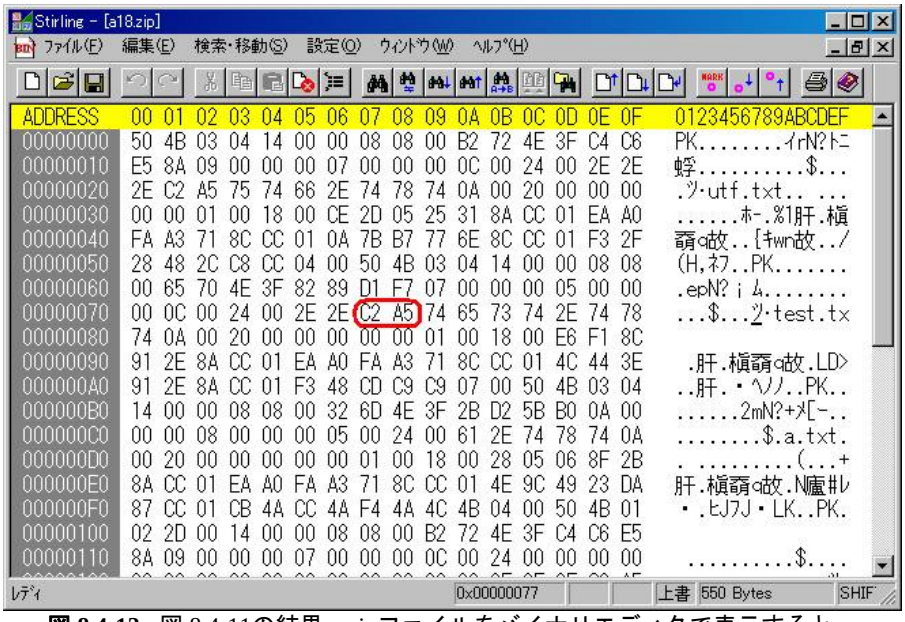

図 8.4-12 : 図 8.4-11の結果。zipファイルをバイナリエディタで表示すると、

「円記号(u00a5)」はそのまま「円記号(UTF-8 の 0xC2 0xA5)」のままだった

8.5. DotNetZipのデフォルト文字コード「IBM437」について

DotNetZipでは、既定の文字コードは「IBM437」という文字コードである。この文字コードがどのような挙動をするかの簡単なサンプル・プログラム(図 8.5-1)である。
図 8.5-2を確認すると「円記号(0xA5 0x00)」が「0x9D」に変換されている。もしこのような挙動をする場合、DotNetZipが(既定の文字コードのままで)使われているかもしれない。

```
using System;
using System.Text;
using System.IO;

public class EncTest{
 public static void Main(string[] args){
  String str = "..¥test.txt あいう";
  Encoding myEncoding1 = Encoding.GetEncoding("utf-16");
  byte[] byteHako1 = myEncoding1.GetBytes(str);
  Console.WriteLine(str + "(UTF-16) =¥r¥n" + BitConverter.ToString(byteHako1));

  Encoding myEncoding2 = Encoding.GetEncoding("IBM437");
  byte[] byteHako2 = myEncoding2.GetBytes(str);
  Console.WriteLine(str + "(IBM437) =¥r¥n" + BitConverter.ToString(byteHako2));

  Encoding myEncoding3 = Encoding.GetEncoding("shift_jis");
  byte[] byteHako3 = myEncoding3.GetBytes(str);
  Console.WriteLine(str + "(Shift-JIS) =¥r¥n" + BitConverter.ToString(byteHako3));

  Encoding myEncoding4 = Encoding.GetEncoding("utf-8");
  byte[] byteHako4 = myEncoding4.GetBytes(str);
  Console.WriteLine(str + "(UTF-8) =¥r¥n" + BitConverter.ToString(byteHako4));
 }
}
```

図 8.5-1 : 文字コード「IBM437」の挙動を確認するプログラム(UNICODE で保存)

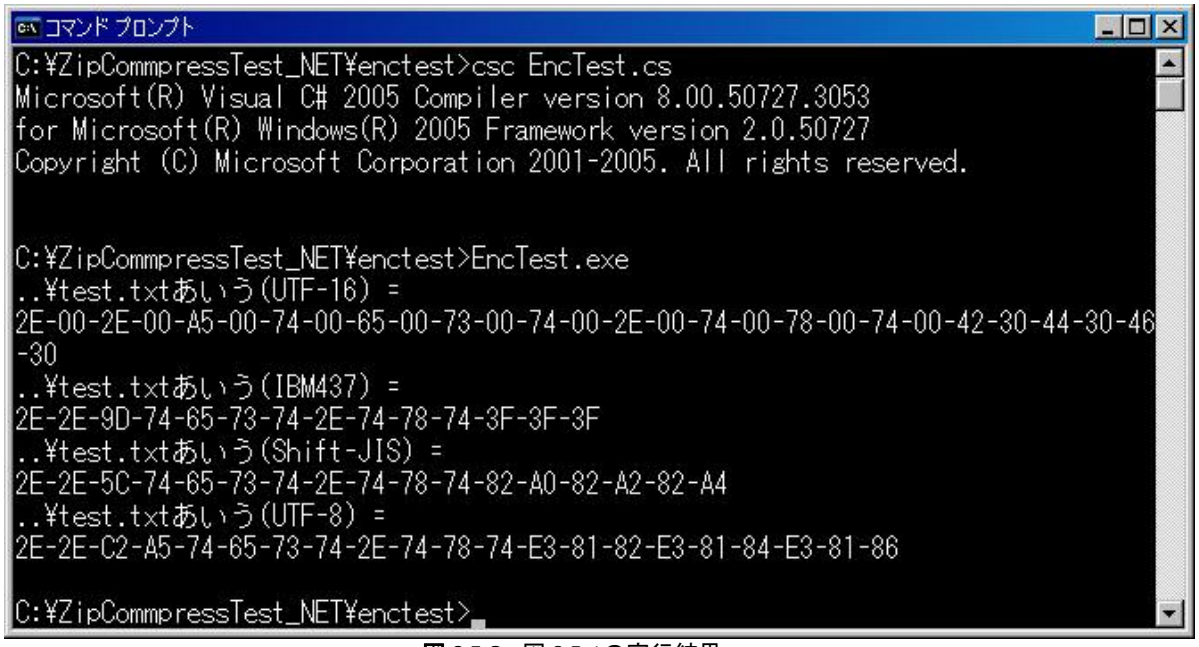

図 8.5-2 : 図 8.5-1の実行結果

9. まとめ

現時点では、MS-Windows の ZIP フォルダなど MS-Windows 上で動作する展開ツールのほとんどが UNICODE に対応していない以上、ZIP ファイル内部に圧縮して保存するファイルのファイル名には、ANSI コードを選択せざるを得ないだろう。
システム設計がこのような場合、Webアプリケーション開発者を含めZIP圧縮を行うアプリケーションに携わっている開発者の方で、UNICODEのファイル名が汚染データ[2]である場合、一旦ファイル名をUNICODEからANSIコードへ変換した後で、サニタイズ処理[3]を実施しなければ、本文書で指摘したようなZIP展開時に想定していないディレクトリへZIP圧縮されたファイルが展開されるだろう。

10. 検証作業者

NTT コミュニケーションズ株式会社
IT マネジメントサービス事業部ネットワークマネジメントサービス部
セキュリティオペレーションセンター
佐名木 智貴
本城 敏信

11. 参考

1. UNICODE とセキュリティ
http://openmya.hacker.jp/hasegawa/public/20041030/unicode-and-security.pdf
2. UTF-8.jp
http://www.utf-8.jp/
3. Unicode とサニタイジング回避テクニック ver1.6
http://rocketeer.dip.jp/secProg/unicodebug007.pdf
4. セキュア Web プログラミング Tips 集(出版社:株式会社ソフト・リサーチ・センター)
ISBN=978-4883732562
5. IPA ISEC セキュアプログラミング講座 ver1 8-1.Windows パス名の落とし穴
http://www.ipa.go.jp/security/awareness/vendor/programmingv1/b08_01.html
6. IPA ISEC セキュアプログラミング講座 ver1 7-7. Unix パス名の安全対策
http://www.ipa.go.jp/security/awareness/vendor/programmingv1/b07_07.html
7. IPA ISEC セキュアプログラミング講座 ver1 8-3. NTFS のセキュリティ機能と落とし穴
http://www.ipa.go.jp/security/awareness/vendor/programmingv1/b08_03.html
8. eo
http://www.softgate.jp/ja/portal/

[2] 汚染データ：入力元が信用できない汚染されているかもしれないデータ
[3] サニタイズ処理：この場合はファイルパスなので、「\」や「/」を含んでいるかどうか、NTFS ストリーム指定があるかどうかなどのバリデーション(入力チェック)を指すだろう

9. Lhaplus
http://www7a.biglobe.ne.jp/~schezo/
10. java.util.zip
http://java.sun.com/javase/ja/6/docs/ja/api/java/util/zip/package-summary.html
11. Apache Ant
http://ant.apache.org/
12. PHP
http://www.php.net/
13. phpMyAdmin
http://www.phpmyadmin.net/home_page/index.php
14. LHA32.DLL for Win32
http://www2.nsknet.or.jp/~micco/mysoft/unlha32.htm
15. LHA for Win32
http://www.asahi-net.or.jp/~gi8s-tkuc/
16. VBScript での zip ファイルの作成
http://kandk.cafe.coocan.jp/jeans/index.php?itemid=234
17. #ziplib（SharpZipLib）を使って ZIP 圧縮、展開（解凍）、リスト表示などを行う
http://dobon.net/vb/dotnet/links/sharpziplib.html
18. .NET Zip Library #ziplib (SharpZipLib)
http://www.icsharpcode.net/OpenSource/SharpZipLib/
19. DotNetZip（Ionic Zip Library）を使って ZIP 書庫を作成する
http://wiki.dobon.net/index.php?.NET%A5%D7%A5%ED%A5%B0%A5%E9%A5%DF%A5%F3%A5%B0%B8%A6%B5%E6%2F93
20. DotNetZip Library
http://dotnetzip.codeplex.com/
21. J#のライブラリを使って ZIP 圧縮、展開（解凍）、リスト表示を行う
http://dobon.net/vb/dotnet/links/createzipfile.html
22. J#の Zip 系クラスは、32bit 限定?
http://d.hatena.ne.jp/Kazzz/20060330/p1
23. Zip 圧縮に J#の Dll を使う時はライセンスに気をつけよう
http://d.hatena.ne.jp/JHashimoto/20110613/1307938219

12. 履歴

- 2010 年 04 月 28 日 ： ver1.0 最初の公開
- 2010 年 10 月 08 日 ： ver1.2 「4.3 php_zip.c (Linux) の場合」、「4.4 zip.lib.php (phpMyAdmin) (Linux) の場合」、「4.5 zip.lib.php (phpMyAdmin) (Linux) の場合その 2」、「5 zipコマンドの場合」、「6 UNLHA32.DLL の場合」と「7 ASP(VBScript) の場合」を新設
- 2011 年 10 月 19 日 ： ver1.3 「8 Microsoft .NET Framework **の場合**」を新設。所属部署の修正。「11 **参考**」にいくつかのリストを追加

13. 最新版の公開URL

http://www.ntt.com/icto/security/data/soc.html#security_report

14. 本レポートに関する問合せ先

NTT コミュニケーションズ株式会社
ソリューションサービス部 第四エンジニアリング部門 セキュリティオペレーション担当

e-mail: scan@ntt.com

以 上