Memcached Injection

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
セキュリティオペレーションセンタ

2009 年 5 月 26 日

Ver. 1.1

1. 調査概要

分散メモリキャッシング・ライブラリである memcached は、使い方によっては、不正行為者がアプリケーションを経由して、memcached プロトコルのコマンドを注入(Injection)することが可能であることを確認した。
適切に memcached ライブラリを用いる方法について、セキュア・プログラミングの観点から検討、検証した結果をここに記す。

- 汚染データ[1]をキャッシュデータとして用いる場合は、本文書の執筆に際して特に注目すべき点を見出すことはできなかった。
- 汚染データをキー名として用いる場合、memcached API によっては、サニタイズ処理を行わないと、memcahed プロトコル上の任意のコマンドが注入され、データ挿入、データ削除などが行われる可能性がある。
- 本文書では、memcached Injection のコンセプトを示すことを目的とした。
  よって memcahed プロトコルの set コマンドに関しての Injection チェックのみを行った。他のコマンドについては、読者諸氏で確認することを推奨する。

技術評論社の「Web+DB Press vol47 (9 参考②)」でも、キーに関してのサニタイズ処理の必要性に関して触れられている箇所がある(9 参考② の P76)。
本文書でセキュア・プログラミングに内容を絞って再度周知を行うことで、memcached を安全に実装したシステムが普及することを願うものである。

2. memcached とは

Danga Interactive によって開発された分散メモリキャッシュ用のデーモンである(9 参考①)。
執筆時点の 1.2.x 系では、通信プロトコルはテキスト(平文)である(9 参考③)。

プロトコルは、非常にシンプルである。
クライアントからのリクエストがあり、それに対してのサーバからのレスポンスがある。この「リクエスト-レスポンス」を一つの単位としている。
基本的に一行につき一コマンドが発行される(デリミタは CrLf である)。
memcached が保持するデータは、データそのものと、そのデータを検索するキーがペアとなっている。
Perl のハッシュ配列や、MS-Windows+COM 上のコレクションオブジェクトと似ている仕組みでデータは保持される。
memcached プロトコルの set コマンドの概要を例として示す(図 4.1-1)。

```
set[SP]キー名[SP]フラグ[SP]有効期限[SP]データのバイト数[CrLf]
データ(CrLf を含み、データを示すバイト数分)[CrLf]
```

図 4.1-1 : memcached **プロトコルの** set **コマンドの書式**([SP]**は半角スペースを示す**)

図 4.1-1 のように、データに関して任意のバイト列が保持できるように考慮されているが、キー名に関しては、そのような考慮がされていない。

[1]汚染データ：不正行為者によって任意の操作が可能なデータ

実際、プロトコル仕様書(9 参考③)にも、キー名はアルファベットのみで 250 文字までと制限されている(図 4.1-2)。
また、エスケープ法についての記載もみつけることができなかった。

```
Keys
----

Data stored by memcached is identified with the help of a key. A key is a
text string which should uniquely identify the data for clients that are
interested in storing and retrieving it.  Currently the length limit of a key
is set at 250 characters (of course, normally clients wouldn't need to use
such long keys); the key must not include control characters or whitespace.
```

図 4.1-2 : memcached プロトコルの仕様書の一部

3. memcahced Injection の概要

「2. memcached とは」で記した通り、memcached プロトコルは、CrLf をデリミタとしている。
よって、memcached Injection とは端的に CrLf Injection そのものである(9 参考⑨、および⑩)。
CrLf を含めたキー名をライブラリ(memcached API)に渡した場合、ライブラリによっては、サニタイズ処理が行われずに、そのまま任意の memcached プロトコルのコマンドが Injection される可能性がある。

4. キー名に関しての検証結果

本章では、memcached にアクセスするための各種ライブラリに対して、改行を含ませたキー名を指定し、どのような結果になるのかを検証した。

4.1. Cache::Memcached (Perl の場合)

4.1.1 検証環境

OS : CentOS5.1
Memcached ver1.2.6
Perl ver5.8.8
libevent ver1.4.8-stable
Perl Time::HiRes ver1.9715
Perl String::CRC32 ver1.4
Perl Storable ver2.18
Apache 2.2.9
ライブラリ
　Cache-Memcached ver1.24

4.1.1 検証結果

検証に使用した Perl のコードは図 4.1-1 である。

```
#! /usr/bin/perl

use strict;
use warnings;
use CGI;
use Cache::Memcached;

my ${reqObj} = new CGI;
my ${keyName} = ${reqObj}->param('keyName');
my ${keyValue} = ${reqObj}->param('keyValue');
my ${keyList} = ${reqObj}->param('keyList');

my ${CMobj} = Cache::Memcached->new({servers=>['127.0.0.1:11211']});

print "Content-Type: text/html¥r¥n¥r¥n";

print "<html><head><title>Perl-Memcached Test</title></head><body>¥n";
print "<form method=¥"post¥" action=¥"¥"><table border=¥"1¥">";
print "<tr><td>keyName<br><textarea name=¥"keyName¥" rows=¥"4¥" cols=¥"30¥">" .
HTMLEscape(${keyName}) . "</textarea></td>¥n";
print "<td>keyValue<br><textarea name=¥"keyValue¥" rows=¥"4¥" cols=¥"30¥">" .
HTMLEscape(${keyValue}) . "</textarea></td></tr>¥n";
print "<tr><td colspan=¥"2¥">keyList<input type=¥"text¥" name=¥"keyList¥" value=¥"" .
HTMLEscape(${keyList}) . "¥" size=¥"72¥">";
print "<input type=¥"submit¥"></td></tr></table>";
print "</form><HR>¥n";

if(0 != length(${keyName}) && 0 != length(${keyValue})){
 ${CMobj}->add(${keyName} => ${keyValue});
 print "data set success!<hr>¥n";
}
my @myList;
my $i;
if(0 != length(${keyList})){
 @myList = split(/,/,${keyList});
 foreach $i(@myList){
  print HTMLEscape($i) . " = " . HTMLEscape(${CMobj}->get($i)) . "<br>¥n";
 }
}

print "</body></html>";

sub HTMLEscape{
    my $str;
    $str = $_[0];
    $str =~ s/¥&/¥&amp¥;/g;
    $str =~ s/¥</¥&lt¥;/g;
    $str =~ s/¥>/¥&gt¥;/g;
    $str =~ s/¥"/¥&quot¥;/g;
    $str =~ s/¥'/¥&¥#39¥;/g;
    return $str;
}
__END__
```

図 4.1-1 : perl で memcached にアクセスする検証コード

まずは、図 4.1-2 で、正常に memcached 上にデータ保持が可能かどうかを確認した。次に、図 4.1-3 のような改行を含むキー名を指定した。その結果が、図 4.1-4 である。この結果からキー名に改行を含められる場合、memcahed プロトコルに対しての Command Injection の危険性があることが確認できる。
実際のプログラムでは、キー名に汚染データが渡される状況は、稀であるとは思われるが、もしキー名に汚染データを渡す可能性がある場合には、改行についてサニタイズ処理する必要がある。

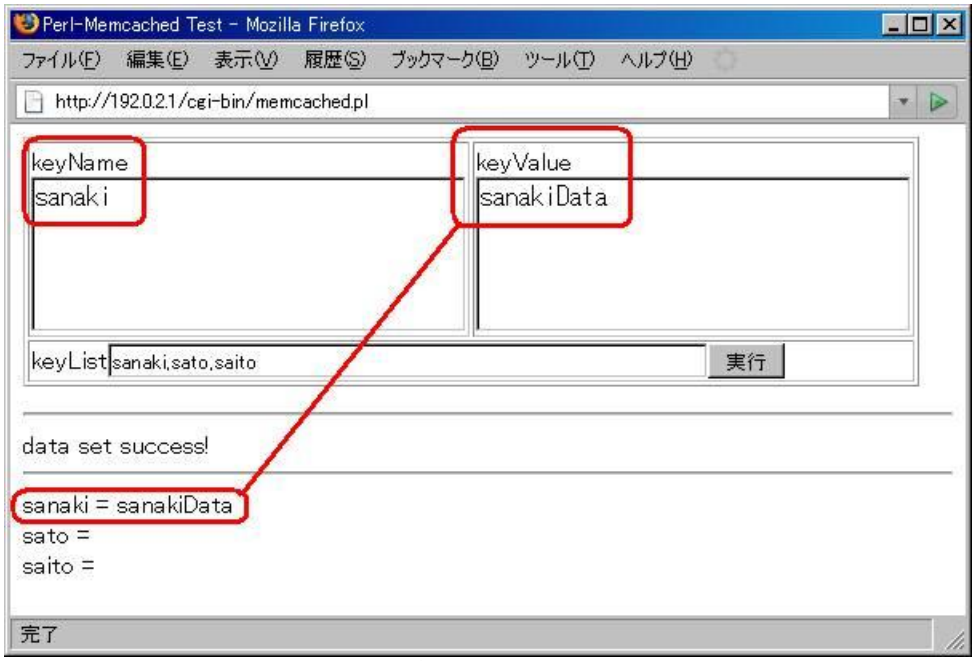

図 4.1-2 : 図 4.1-1 を使って、データ書き込みを行い、正常動作を確認した

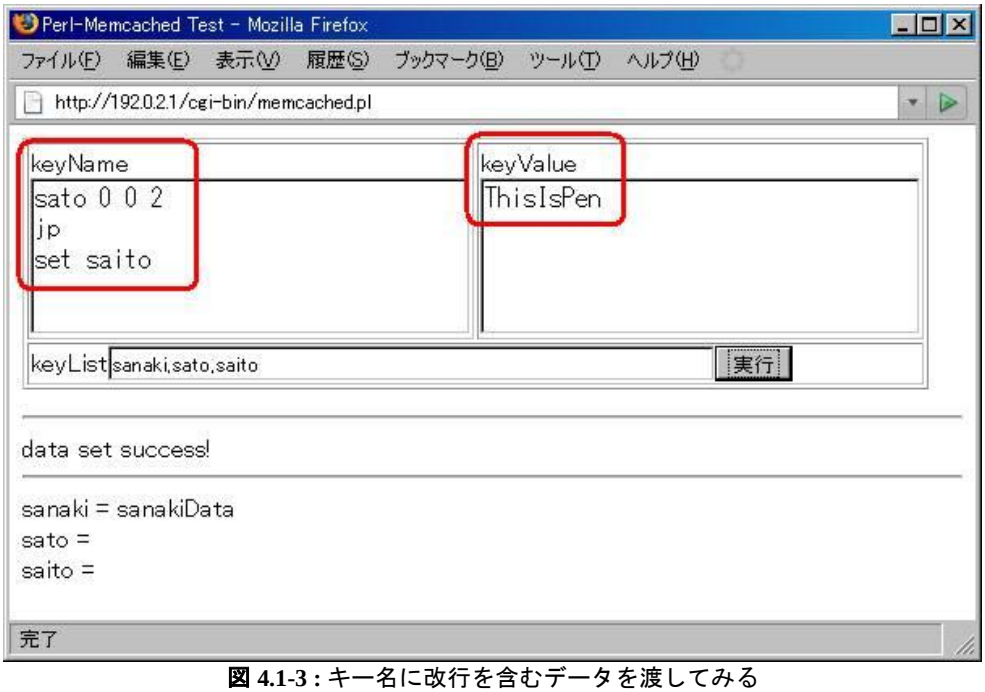

図 4.1-3 : キー名に改行を含むデータを渡してみる

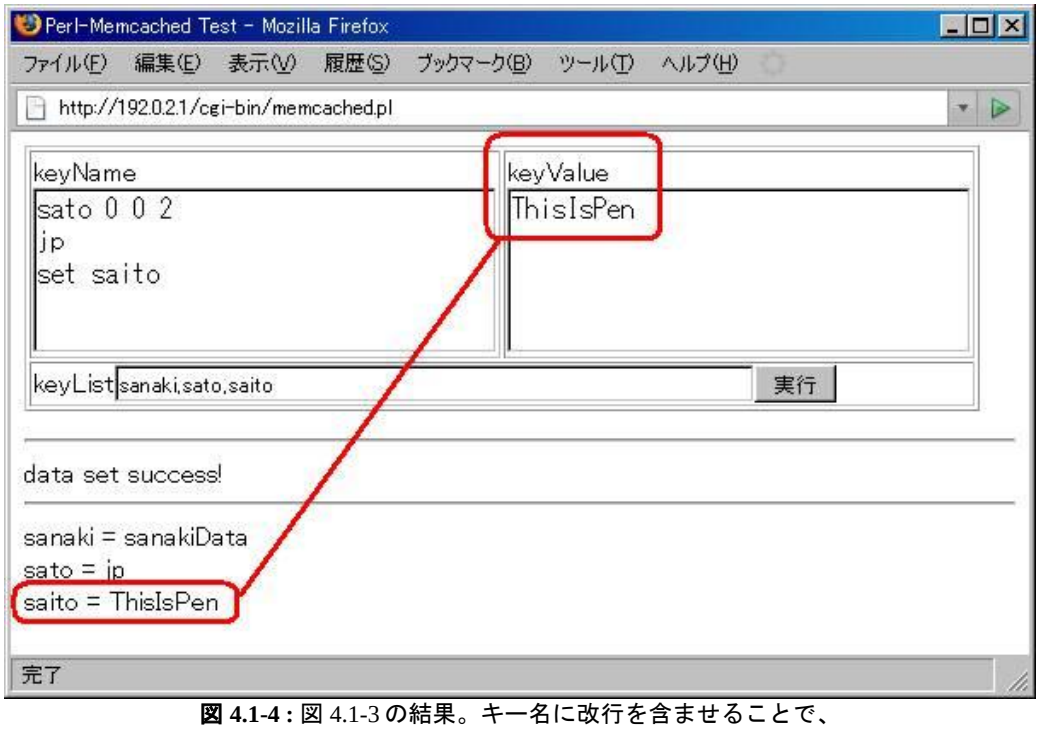

図 4.1-4 : 図 4.1-3 の結果。キー名に改行を含ませることで、
memcached プロトコルへの Command Injection が成功していることが確認できる

4.2. Memcached client for java (Java の場合)

4.2.1 検証環境

OS：CentOS5.1
Memcached ver1.2.6
JDK ver1.5.0_16
libevent ver1.4.8-stable
Java logging-log4j ver1.2.9
Tomcat 5.5.26
ライブラリ
　Memcached client for java ver2.0.1 (JDK5 Compiled)

4.2.2 検証結果

検証に使用した Java(JSP)のコードは図 4.2-1 である。

```
<%@ page import="com.danga.MemCached.*" %>
<%!
 public String myPrintOut(String iKeyName,MemCachedClient mc){
  String str = (String)mc.get(iKeyName);
  return HTMLEscape(iKeyName) + " = " + HTMLEscape(str) + "<br>¥n";
 }
 public String HTMLEscape(String iStr){
  if(iStr == null || iStr.equals("")){
    return "";
  }else{
    String ansStr = iStr.replaceAll("&","&amp;");
    ansStr = ansStr.replaceAll("'","&#39;");
    ansStr = ansStr.replaceAll("¥"","&quot;");
    ansStr = ansStr.replaceAll("<","&lt;");
    return ansStr.replaceAll(">","&gt;");
  }
 }
%>
<%
 String keyName = request.getParameter("keyName");
 String keyValue = request.getParameter("keyValue");
 if(keyName == null){
  keyName = "";
 }
 if(keyValue == null){
  keyValue = "";
 }
%>
<html>
 <head>
  <title>Java - MemCached Test</title>
 </head>
 <body>
  <form action="" method="post">
   <table border="1">
    <tr>
     <td>keyName<br><textarea name="keyName" rows="4"
cols="32"><%=HTMLEscape(keyName)%></textarea></td>
     <td>keyValue<br><textarea name="keyValue" rows="4"
cols="32"><%=HTMLEscape(keyValue)%></textarea></td>
    </tr></table>
   <input type="submit">
  </form><hr>
<%
 SockIOPool pool = SockIOPool.getInstance();
 pool.setServers(new String[]{"localhost:11211"});
 pool.initialize();


 MemCachedClient mc = new MemCachedClient();
 if(0 < keyName.length() && 0 < keyValue.length()){
  mc.set(keyName,keyValue);
  out.println("Data set success!");
 }
out.println("<hr>");
out.println(myPrintOut("sanaki",mc));
out.println(myPrintOut("sato",mc));
out.println(myPrintOut("saito",mc));
%>
 </body></html>
```

図 4.2-1 : Java で memcached にアクセスする検証コード

まずは、図 4.2-2 で、正常に memcached 上にデータ保持が可能かどうかを確認した。次に、図 4.2-3 のような改行を含むキー名を指定した。その結果が、図 4.2-4 である。
Memcached プロトコルに対しての Injection は失敗に終わっている。その後、図 4.2-5～図 4.2-7 までの操作で、Memcached Client for java では、キー名を URL エンコードしていることを確認した。
よって、Memcached Client for java を使ったシステムでは、キー名に対しての memcached Injection はできないことが確認された。

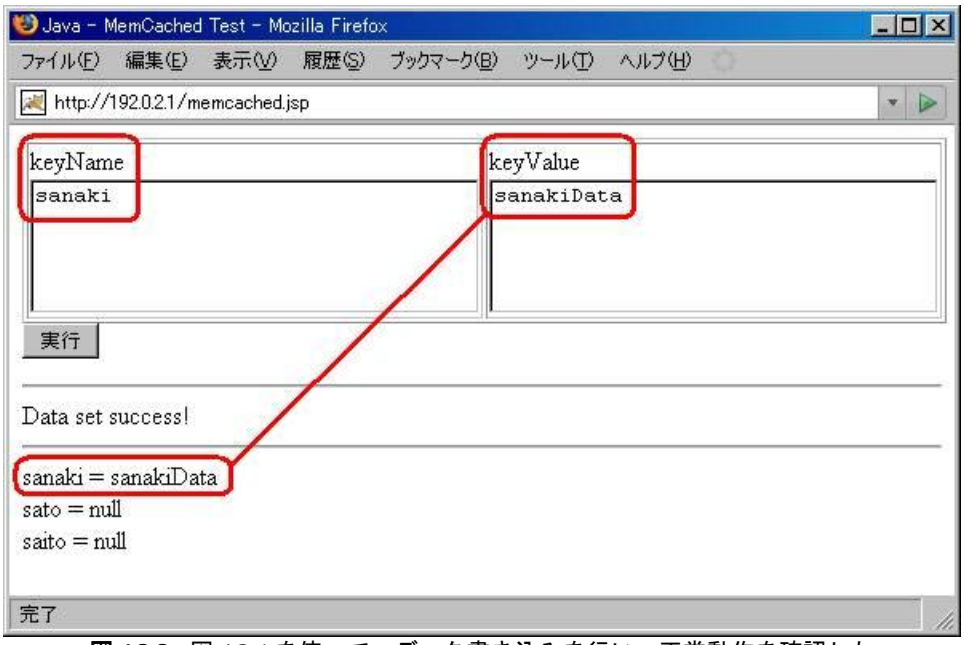

図 4.2-2 : 図 4.2-1 を使って、データ書き込みを行い、正常動作を確認した

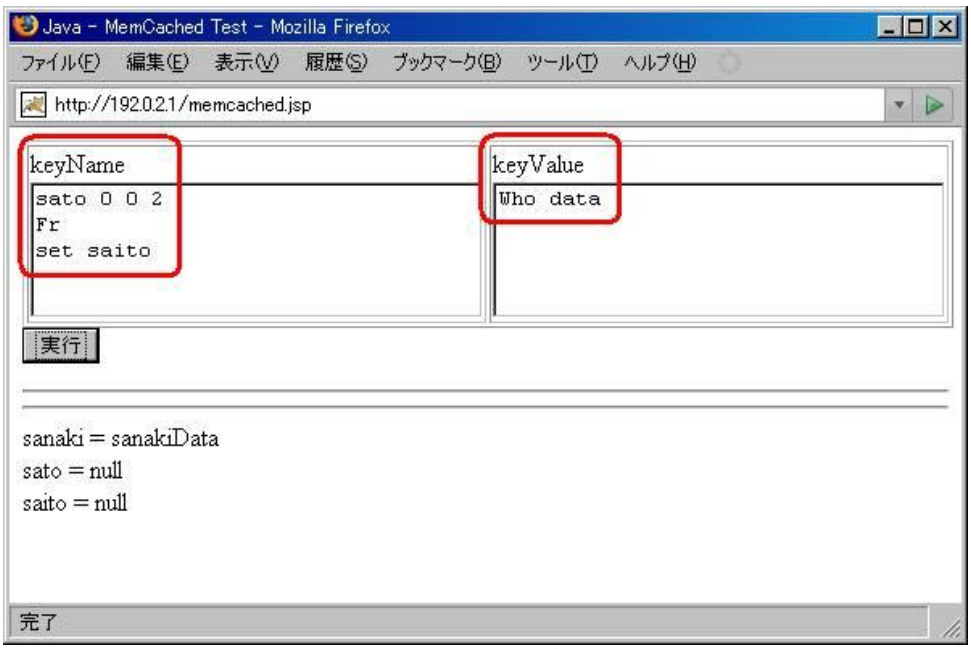

図 4.2-3 : キー名に改行を含むデータを渡してみる

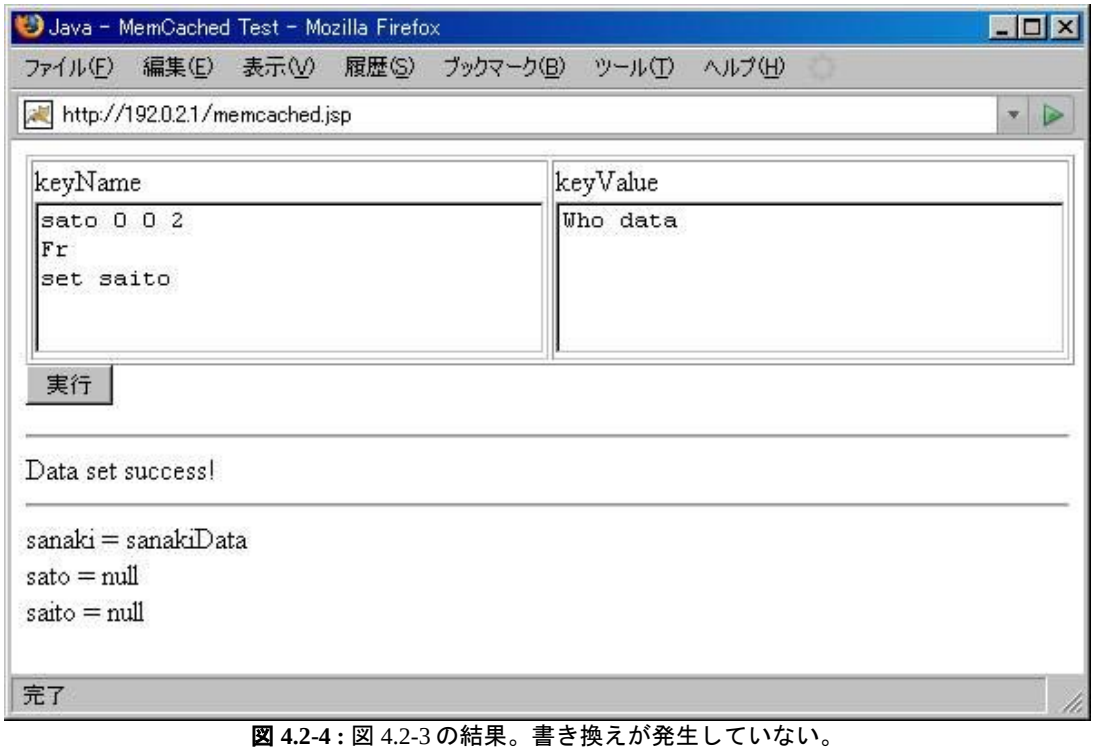

図 4.2-4 : 図 4.2-3 の結果。書き換えが発生していない。

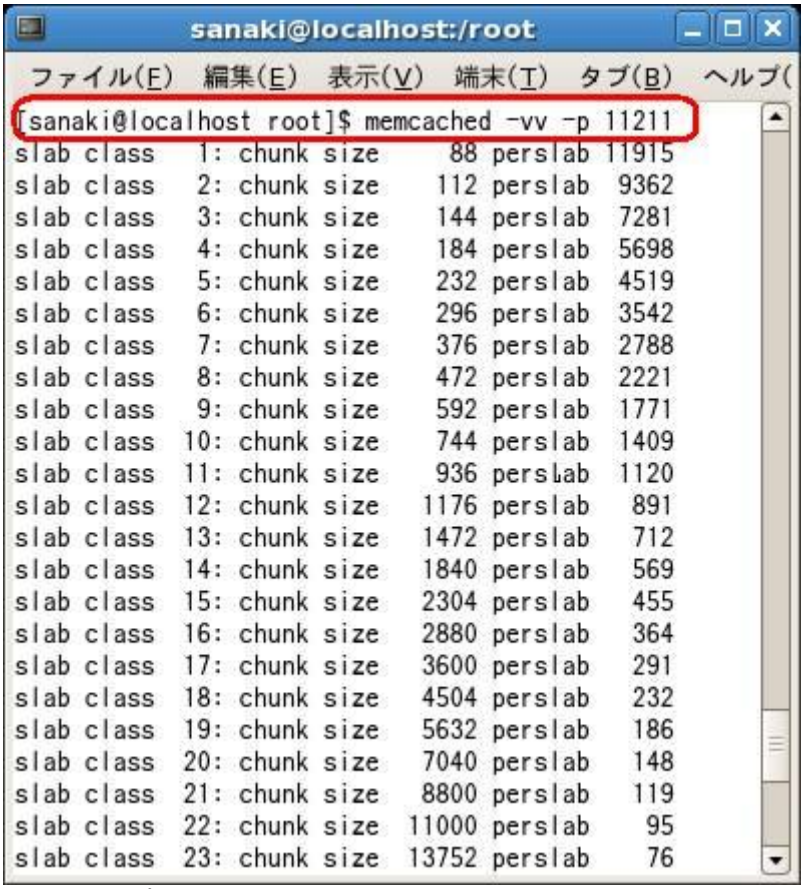

図 4.2-5 : 「-vv」オプション付きで memcached を起動することで、状況確認が可能である

図 4.2-6 : 図 4.2-3～図 4.2-4 の memcachd コンソール側の内容。

URL エンコードされていることが確認できる

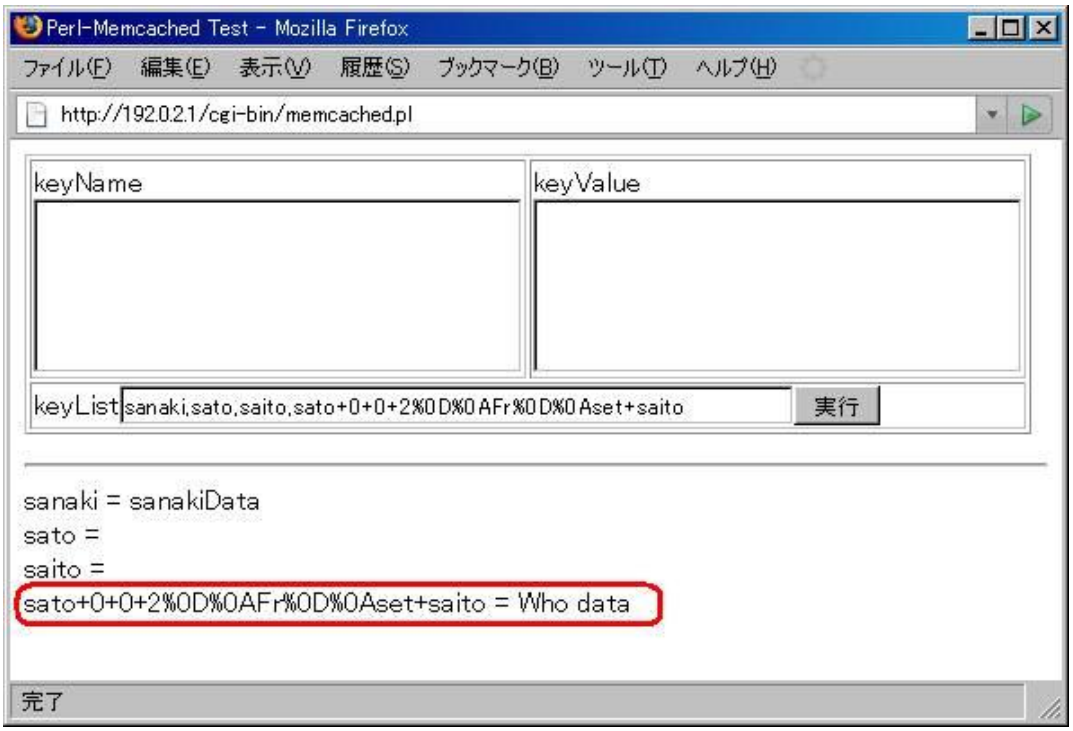

図 4.2-7 : 図 4.2-6 後、Perl を使って情報を取得した結果、

キー名は URL エンコードされていることが確認できる

4.3. libmemcached (C の場合)

4.3.1 検証環境

OS：CentOS5.1
Memcached ver1.2.6
gcc ver4.1.2 20070626
libevent ver1.4.8-stable
Apache ver2.2.9
ライブラリ
　libmemcached ver0.23

4.3.2 検証結果

検証に使用した C(CGI)のコードは図 4.3-1 である。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "libmemcached/memcached.h"

void htmlescape(char *inStr){
  char *p;
  p = inStr;
  while(*p != '¥0'){
    switch(*p){
    case '&':
      printf("&amp;");  break;
    case '¥"':
      printf("&quot;");  break;
    case '¥'':
      printf("&#39;");  break;
    case '<':
      printf("&lt;");  break;
    case '>':
      printf("&gt;");  break;
    default:
      printf("%c",*p);  break;
    }
    p++;
  }
}

int Hex2Int(char hex){
  int myHexIntHash[32] = {-1,10,11,12,13,14,15,-1,-1,-1,-1,-1,-1,-1,-1,-1,0,1,2,3,4,5,6,7,8,9,-1,-1,-
1,-1,-1,-1};
  return myHexIntHash[hex & 31];
}

void urldecode(char* inStr){
  signed long sl;
  char c;
  char *p, *q;
  if(inStr != NULL){
```

```
    p = inStr;
    q = inStr;
    while(*p != '¥0'){
      switch(*p){
      case '%':
        sl = (signed long)(16*Hex2Int(*(p+1)) + Hex2Int(*(p+2)));
        c = (char)sl;
        *q = c;
        q++;  p+=3;  break;
      case '+':
        *q = ' ';
        p++;  q++;  break;
      default:
        if(p != q){  *q = *p;  }
        p++;  q++;  break;
      }
    }
    *q = '¥0';
  }
}

int main(int argc,char* argv[]){
  struct memcached_st *mc;
  struct memcached_server_st *servers;
  memcached_return rmc;
  char *ret,*q, *p, *keyName, *keyValue;
  long qLen;
  keyName = NULL;
  keyValue = NULL;
  q = NULL;
  ret = NULL;
  ret = getenv("QUERY_STRING");
  if(ret != NULL){
    qLen = strlen(ret) + 1;
    q = malloc(qLen);
    memset(q,0x00,qLen);
    memcpy(q,ret,qLen);
    p = q;
    while(*p != '¥0'){
      if(strncmp(p,"keyName",7) == 0){
        p+=8;
        keyName = p;
        while(*p != '¥0'){
          if(*p == '&'){  *p = '¥0';  }else{  p++;  }
        }
        p++;
      }else if(strncmp(p,"keyValue",8) == 0){
        p+=9;
        keyValue = p;
        while(*p != '¥0'){
          if(*p == '&'){  *p = '¥0';  }else{  p++;  }
        }
        p++;
      }else{  p++;  }
    }
  }
  urldecode(keyName);
  urldecode(keyValue);
  printf("Content-Type: text/html¥r¥n¥r¥n");
  printf("<html><head><title>libmemcached Test</title></head><body>¥n");
  mc = memcached_create(NULL);
  servers = memcached_servers_parse("127.0.0.1:11211");
  rmc = memcached_server_push(mc,servers);
```

```
  if(rmc == MEMCACHED_SUCCESS){
    if(keyName != NULL && keyValue != NULL){
      rmc = memcached_set(mc,keyName,strlen(keyName),keyValue,strlen(keyValue),0,0);
    }
    printf("<form method=¥"get¥"><table border=¥"1¥">");
    printf("<tr><td>keyName<br><textarea name=¥"keyName¥" rows=¥"4¥" cols=¥"30¥">");
    if(keyName != NULL){  htmlescape(keyName);  }
    printf("</textarea></td><td>keyVakue<br><textarea name=¥"keyValue¥" rows=¥"4¥" cols=¥"30¥">");
    if(keyValue != NULL){  htmlescape(keyValue);  }
    printf("</textarea></td></tr></table><input type=¥"submit¥"></form><hr></body></html>");
  }
  if(q != NULL){  free(q);  }
  memcached_free(mc);
  memcached_server_list_free(servers);
  return 0;
}
```

図 4.3-1 : C で memcached にアクセスする検証コード

コンパイルは「gcc –lmemcached -o memcached.cgi memcached.c」である

まずは、図 4.3-2～図 4.3-3 で、正常に memcached 上にデータ保持が可能かどうかを確認した。次に、図 4.3-4 のような改行を含むキー名を指定した。その結果が、図 4.3-5 である。この結果からキー名に改行を含ませることができるような場合、memcahed プロトコルに対しての Command Injection の危険性があるということが確認された。
実際の開発現場では、キー名に汚染データが渡される状況は、稀であるとは思われるが、もしキー名に汚染データを渡す場合には、改行についてサニタイズ処理する必要がある、ということである。

図 4.3-2 : 図 4.3-1 を使って、データ書き込みを行い、正常動作を確認した

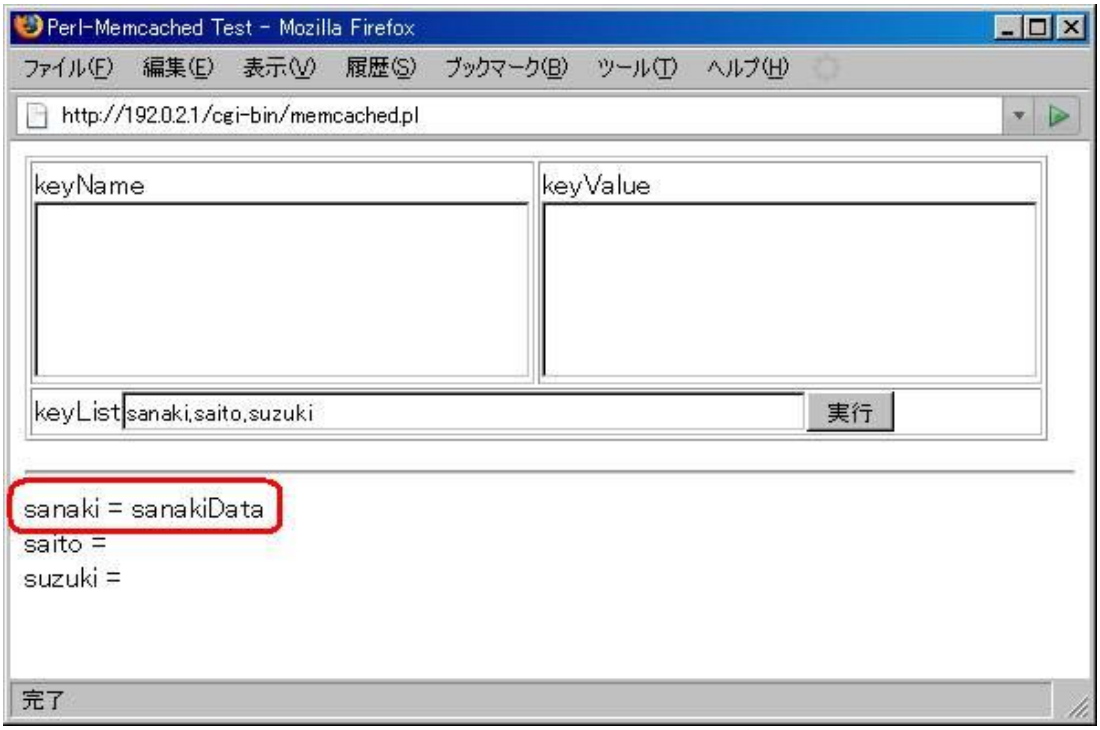

図 4.3-3 : 図 4.3-2 の結果。Memcached が保持しているデータの表示には、Perl の検証コードを使うことにした。

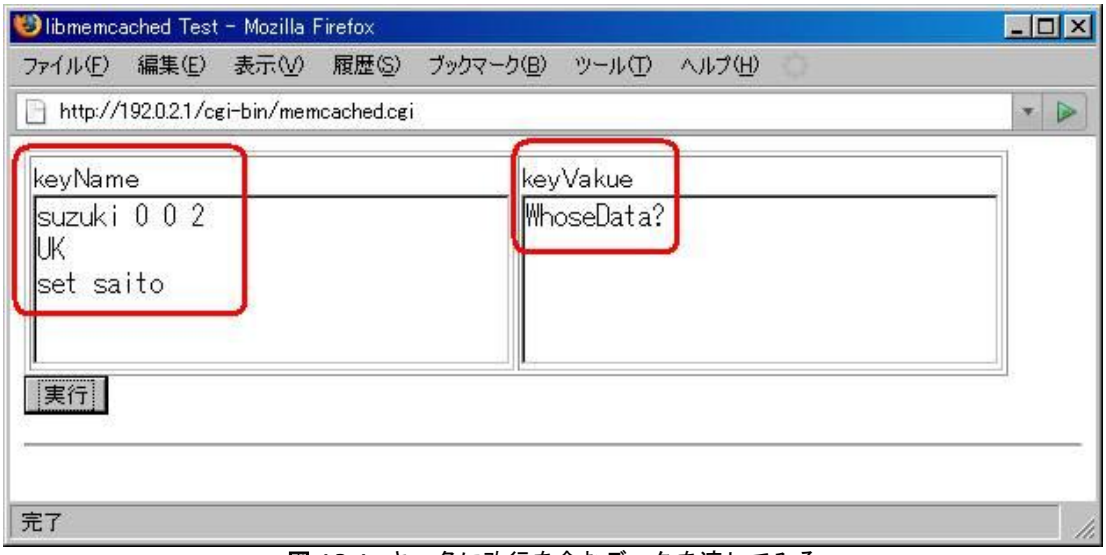

図 4.3-4 : キー名に改行を含むデータを渡してみる

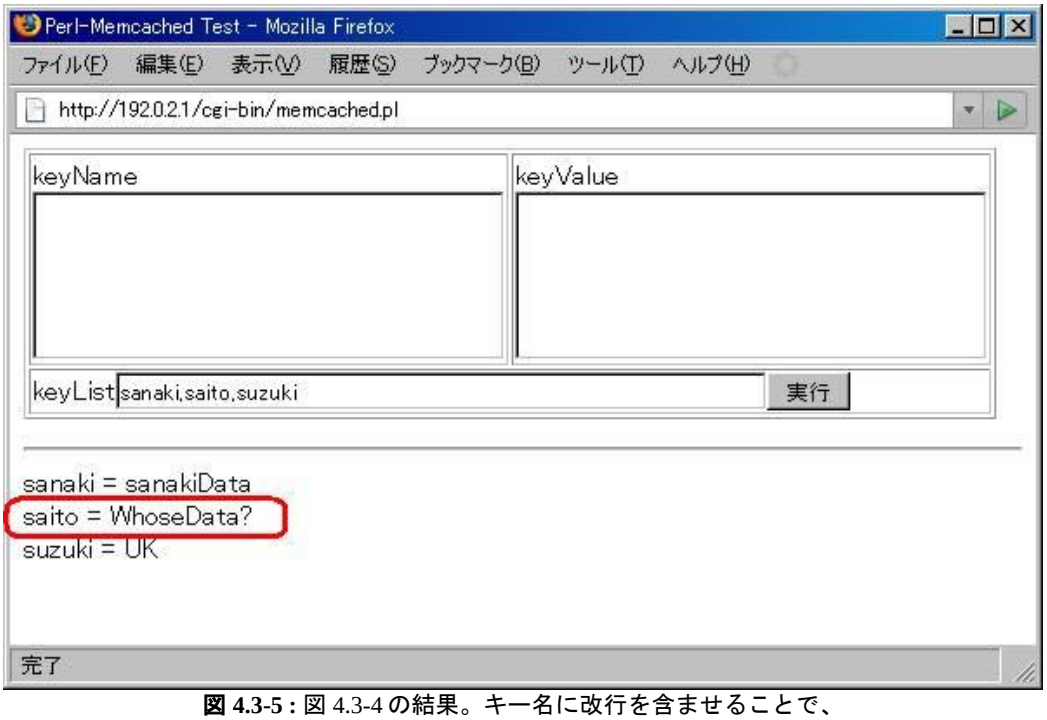

図 4.3-5 : 図 4.3-4 の結果。キー名に改行を含ませることで、
memcached プロトコルへの Command Injection が成功していることが確認できる

4.4. enyim.com Memcached Client (ASP.NET/C#の場合その 1)

4.4.1 検証環境

OS：CentOS5.1 Memcached ver1.2.6
--
OS：Microsoft-WindowsServer2008 & IIS7.0
Microsoft-VisualStudio2008 ExpressEdition ServicePack1
Microsoft .Net Framework 2.0.50727
ライブラリ
　enyim.com Memcached Client ver1.2.0.2

4.4.2 検証結果

検証に使用した ASP.NET2.0/C#のコードは図 4.4-1～図 4.4-3 である。

```
using System;
using System.Text;
using Enyim.Caching;

namespace MemCachedTest_CS{
    public partial class _Default : System.Web.UI.Page{
        protected void Button1_Click(object sender, EventArgs e){
            Enyim.Caching.MemcachedClient myMCC;
            string keyName;
            string keyValue;
            String str;
            object myObj;
            myMCC = new MemcachedClient();
            keyName = TextBox1.Text;
            keyValue = TextBox2.Text;
            if (keyName.Length != 0 && keyValue.Length != 0) {
                myMCC.Store(Enyim.Caching.Memcached.StoreMode.Set,keyName,
Encoding.Unicode.GetBytes(keyValue),0,2*keyValue.Length);
                myObj = myMCC.Get(keyName);
                if (myObj != null){
                    str = Encoding.Unicode.GetString((byte[])myObj);
                    Label3.Text = "<pre>abcdE<br>" + Server.HtmlEncode(keyName) + "=" +
Server.HtmlEncode(str) + "</pre>";
                }else {
                    Label3.Text = "NoData";
                }
            }
        }
    }
}
```

図 4.4-1 : C#で memcached にアクセスする検証コード(C#)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="MemCachedTest_CS._Default" validateRequest="false" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server"><title>Memcached Test</title></head>
<body><form id="form1" runat="server"><div style="width: 248px">
        <asp:Table ID="Table1" runat="server" BorderWidth="1px" Height="161px" Width="362px">
            <asp:TableRow runat="server" BorderWidth="11px">
                <asp:TableCell runat="server" BorderWidth="1px">
                    <asp:Label ID="Label1" runat="server" Text="keyName"></asp:Label>
                </asp:TableCell>
                <asp:TableCell runat="server" BorderWidth="1px">
                    <asp:Label ID="Label2" runat="server" Text="keyValue"></asp:Label>
                </asp:TableCell>
            </asp:TableRow>
            <asp:TableRow runat="server" BorderWidth="11px">
                <asp:TableCell runat="server" BorderWidth="1px">
<asp:TextBox ID="TextBox1" runat="server" Height="72px" TextMode="MultiLine" Width="247px">
</asp:TextBox>
       </asp:TableCell><asp:TableCell runat="server" BorderWidth="1px">
<asp:TextBox ID="TextBox2" runat="server" Height="72px" TextMode="MultiLine" Width="247px">
</asp:TextBox>
                </asp:TableCell>
            </asp:TableRow>
        </asp:Table>
        <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
        <hr />
        <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
</div></form></body></html>
```

図 4.4-2 : C#で memcached にアクセスする検証コード(ASP.NET)

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <sectionGroup name="enyim.com">
<section name="memcached" type="Enyim.Caching.Configuration.MemcachedClientSection, Enyim.Caching"/>
    </sectionGroup>
  </configSections>
  <enyim.com>
    <!--<memcached enabled="true">-->
    <memcached>
      <!-- keyTransformer="" -->
      <servers>
        <add address="192.0.2.1" port="11211"/>
      </servers>
<socketPool minPoolSize="10" maxPoolSize="100" connectionTimeout="00:10:00" deadTimeout="00:02:00"/>
    </memcached>
  </enyim.com>
  <appSettings/>  <connectionStrings/>
  <system.web>
    <compilation debug="true"></compilation>
    <authentication mode="Windows"/>
  </system.web>
  <system.codedom></system.codedom>
  <system.webServer></system.webServer>
</configuration>
```

図 4.4-3 : C#で memcached にアクセスする検証コード(Web.config)

まずは、図 4.4-4～図 4.4-6 で、正常に memcached 上にデータ保持が可能かどうかを確認した。次に、図 4.4-7 のような改行を含むキー名を指定した。その結果が、図 4.4-8 であった。図 4.4-9 は Memcached のログであるが、図 4.4-7～図 4.4-8 の操作時の通信が発生していないことから、改行コードなどを含むキー名を memcached プロトコル上に流さないようにサニタイズ処理しているものと推定できる。

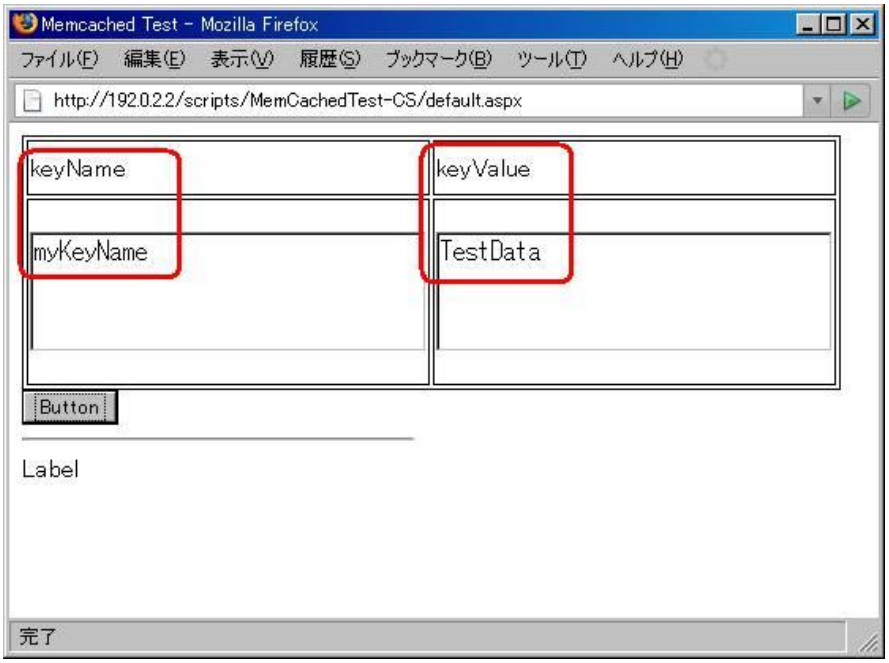

図 4.4-4 : 図 4.4-1 図 4.4-3 を使って、データ書き込みを行い、正常動作を確認した

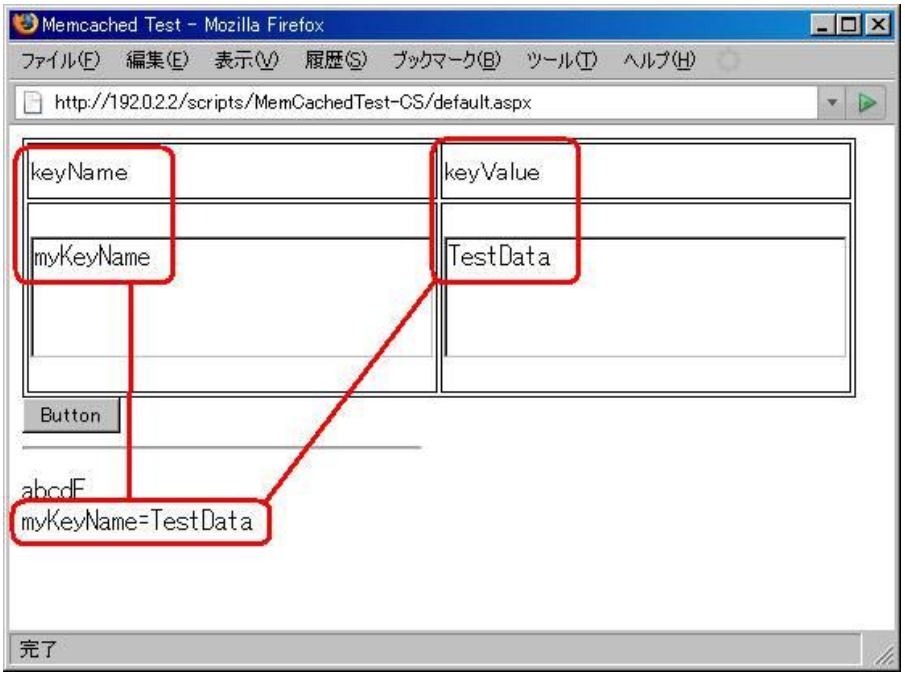

図 4.4-5 : 図 4.4-4 の結果 1

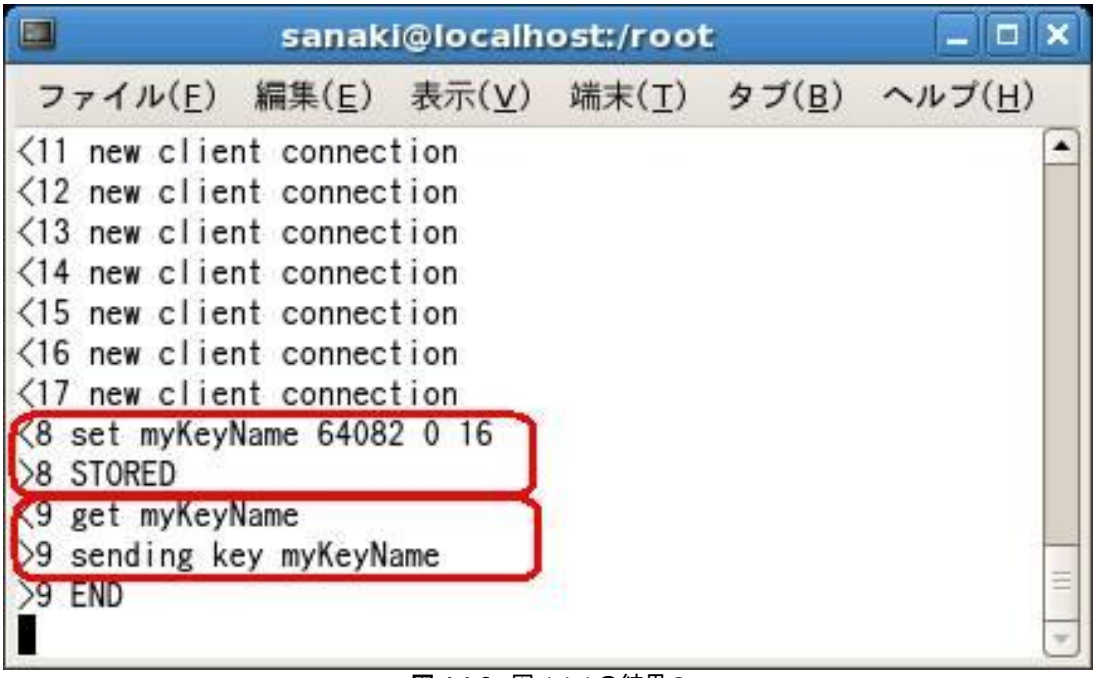

図 4.4-6 : 図 4.4-4 の結果 2

memcached に書き込みが成功している

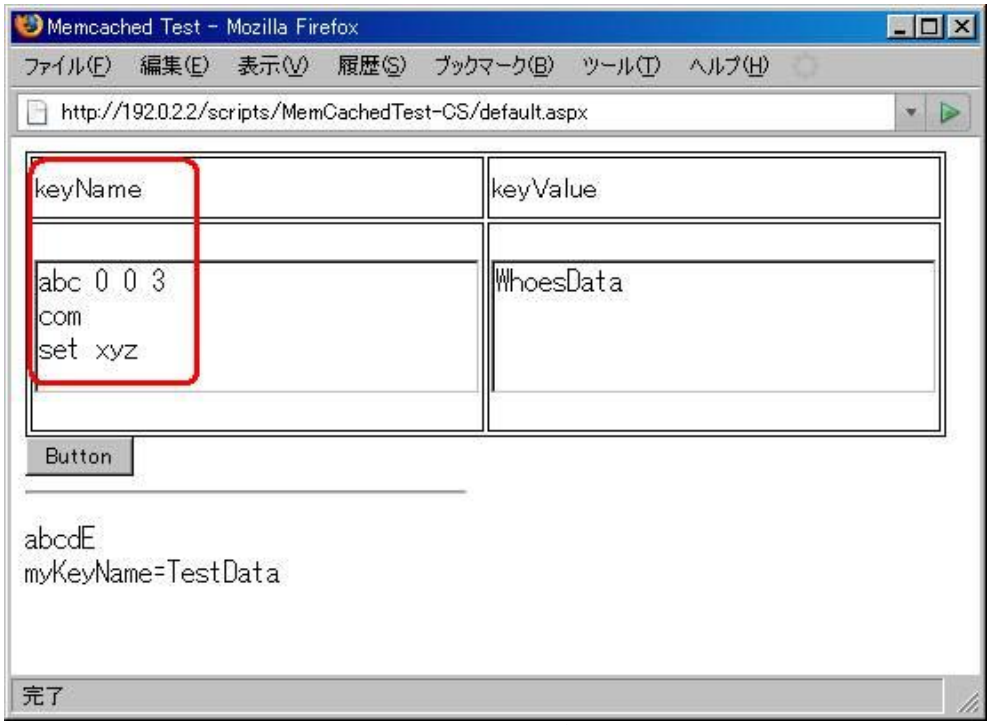

図 4.4-7 : キー名に改行を含むデータを渡してみる

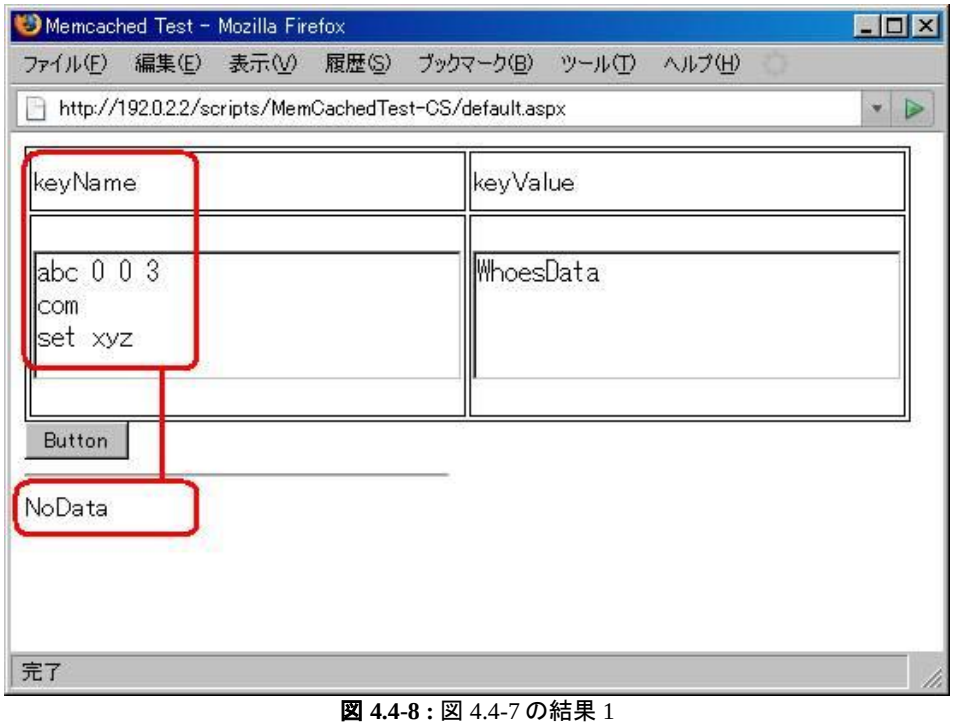

図 4.4-8 : 図 4.4-7 の結果 1

データの取得もできていない状態である

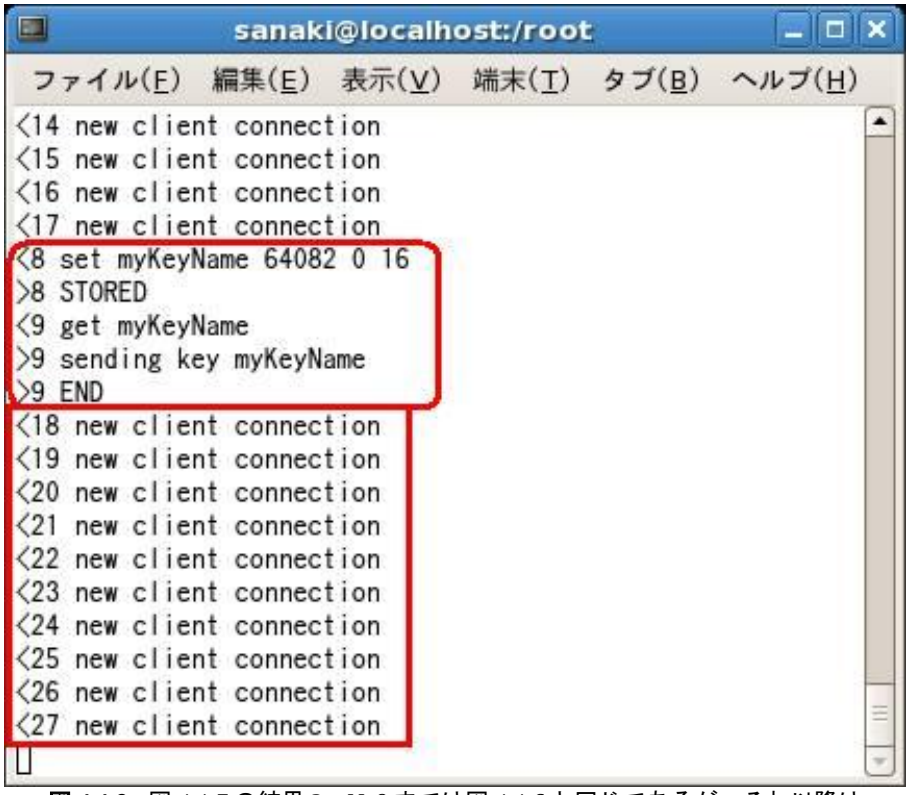

図 4.4-9 : 図 4.4-7 の結果 2。No9 までは図 4.4-6 と同じであるが、それ以降は

そもそも書き込み/読み出しの通信が発生していないことが確認できる

4.5. memcacheddotnet_clientlib (ASP.NET/C#の場合その 2)

4.5.1 検証環境

OS：CentOS5.1 Memcached ver1.2.6
--
OS：Microsoft-WindowsServer2008 & IIS7.0
Microsoft-VisualStudio2008 ExpressEdition ServicePack1
Microsoft .Net Framework 2.0.50727
ライブラリ
　memcacheddotnet_clientlib ver 1.1.5

4.5.2 検証結果

検証に使用した ASP.NET2.0/C#のコードは図 4.5-1 である。

```
using System;
using Memcached.ClientLibrary;

namespace MemCachedTest_CS{
    public partial class _Default : System.Web.UI.Page{
        protected void Button1_Click(object sender, EventArgs e){
            string keyName;
            string keyValue;
            string str;
            string[] memServers = {"172.20.0.65:11211"};
            SockIOPool pool = SockIOPool.GetInstance();
            pool.SetServers(memServers);
            pool.InitConnections = 3;
            pool.MinConnections = 3;
            pool.MaxConnections = 5;
            pool.SocketConnectTimeout = 1000;
            pool.SocketTimeout = 3000;
            pool.MaintenanceSleep = 30;
            pool.Failover = true;
            pool.Nagle = false;
            pool.Initialize();
            MemcachedClient mc = new MemcachedClient();
            mc.EnableCompression = false;
            keyName = TextBox1.Text;
            keyValue = TextBox2.Text;
            if (keyName.Length != 0 && keyValue.Length != 0) {
                mc.Set(keyName, keyValue);
                str = (string)mc.Get(keyName);
                if (str != null){
Label3.Text = "<pre>" + Server.HtmlEncode(keyName) + "=" + Server.HtmlEncode(str) + "</pre>";
                }else {
Label3.Text = "<pre>" + Server.HtmlEncode(keyName) + " is NoData</pre>";
                }
            }
        }
    }
}
```

図 4.5-1：C#で memcached にアクセスする検証コード(C#)

ASP.NET のコードは図 4.4-2 と同一なので省略する。
Web.comfig は、修正することなく既定のまま使用したので、省略する。

まずは、図 4.5-2～図 4.5-3 で、正常に memcached 上にデータ保持が可能かどうかを確認した。
次に、図 4.5-4 のような改行を含むキー名を指定した。その結果が、図 4.5-5 であった。Perl の検証で使用した CGI を使ってアクセスした結果、memcached Injection 可能であることが確認された(図 4.5-6)。また、memcached のコンソールからも、memcached Injection 可能であることが確認された(図 4.5-7)。
つまり、キー名に改行を含ませることができるような場合、memcahed プロトコルに対しての Command Injection の危険性がある。
実際の開発現場では、キー名に汚染データが渡される状況は、稀であるとは思われるが、もしキー名に汚染データを渡す場合には、改行についてサニタイズ処理する必要がある、ということである。

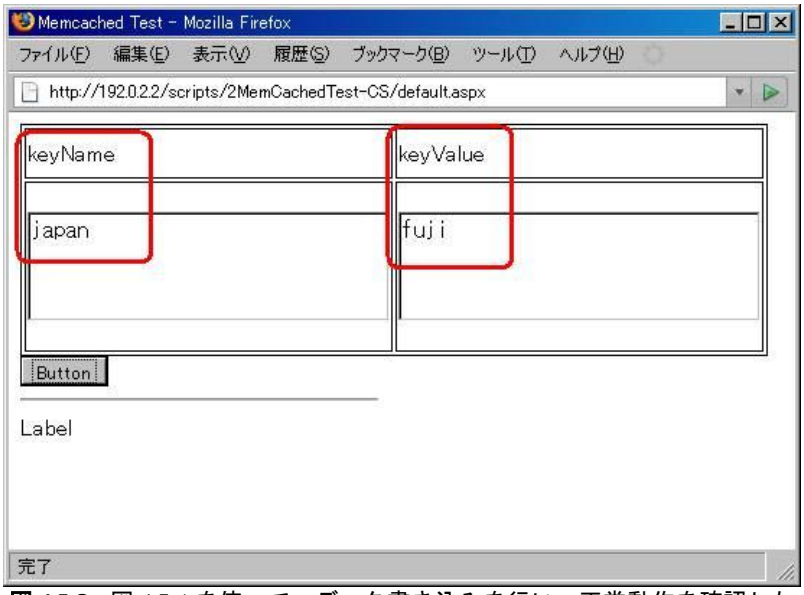

図 4.5-2 : 図 4.5-1 を使って、データ書き込みを行い、正常動作を確認した

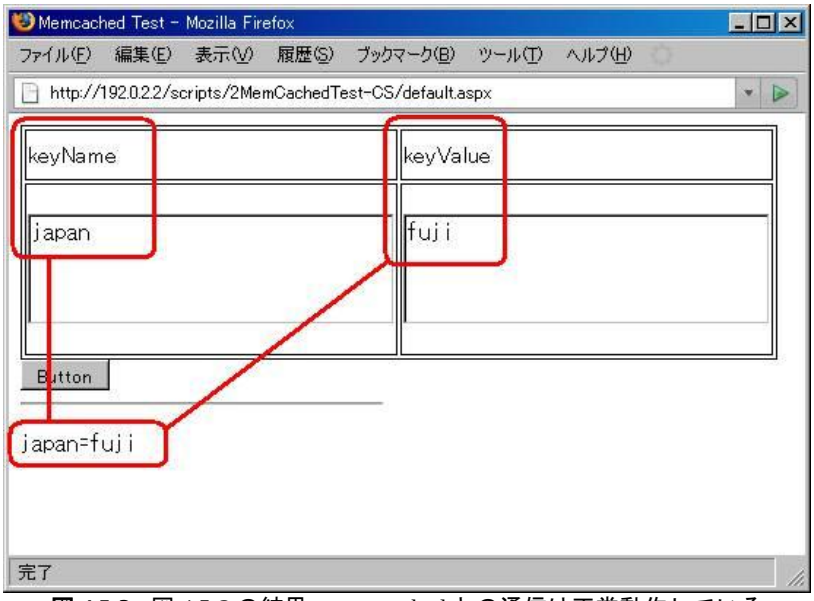

図 4.5-3 : 図 4.5-2 の結果。memcached との通信は正常動作している

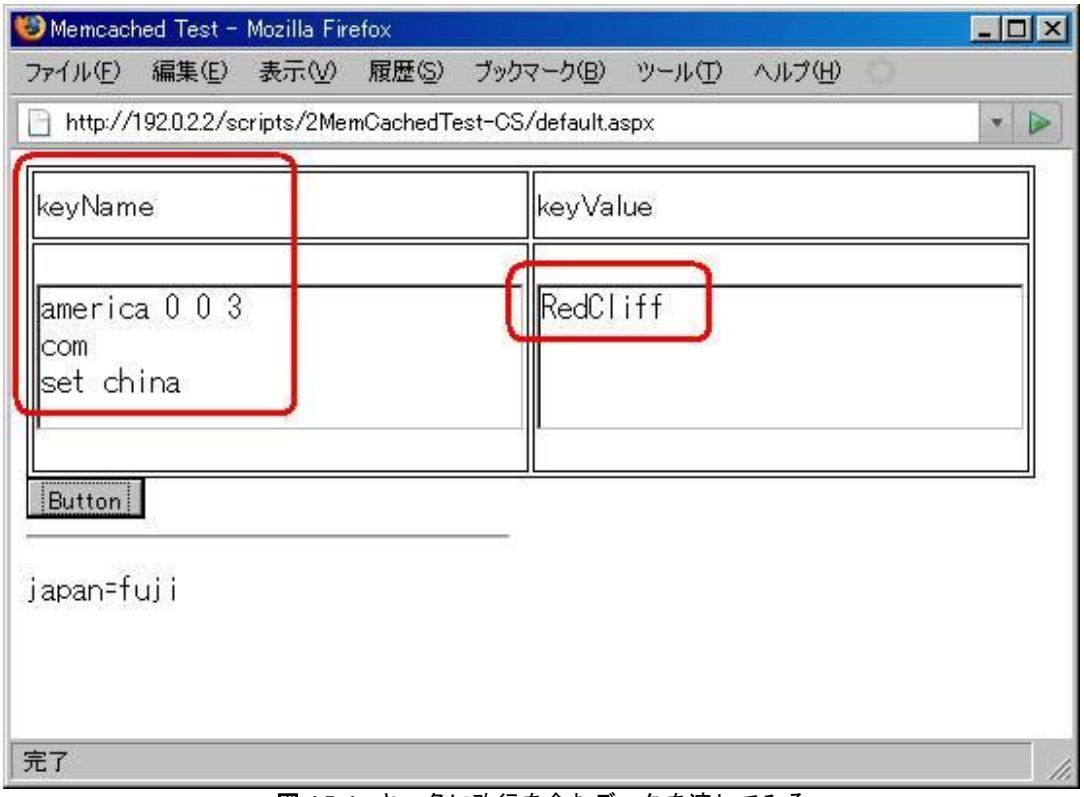

図 4.5-4 : キー名に改行を含むデータを渡してみる

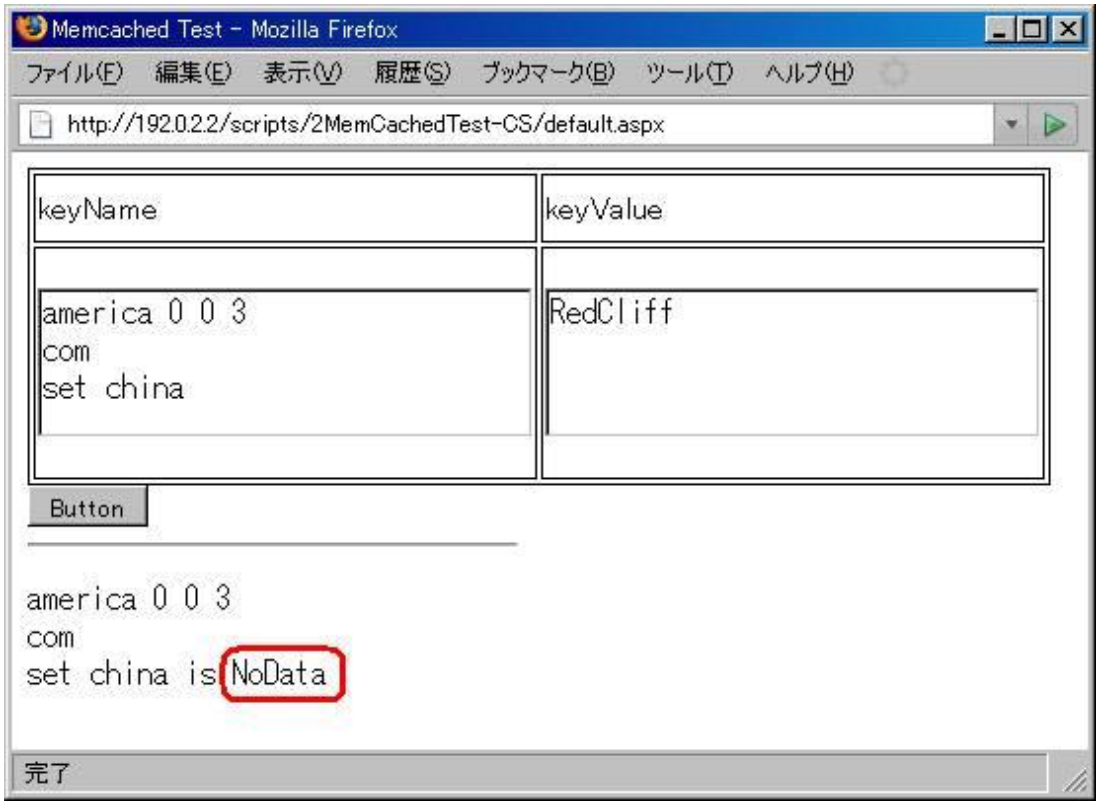

図 4.5-5 : 図 4.5-4 の結果 2。memcached と通信しているように見える

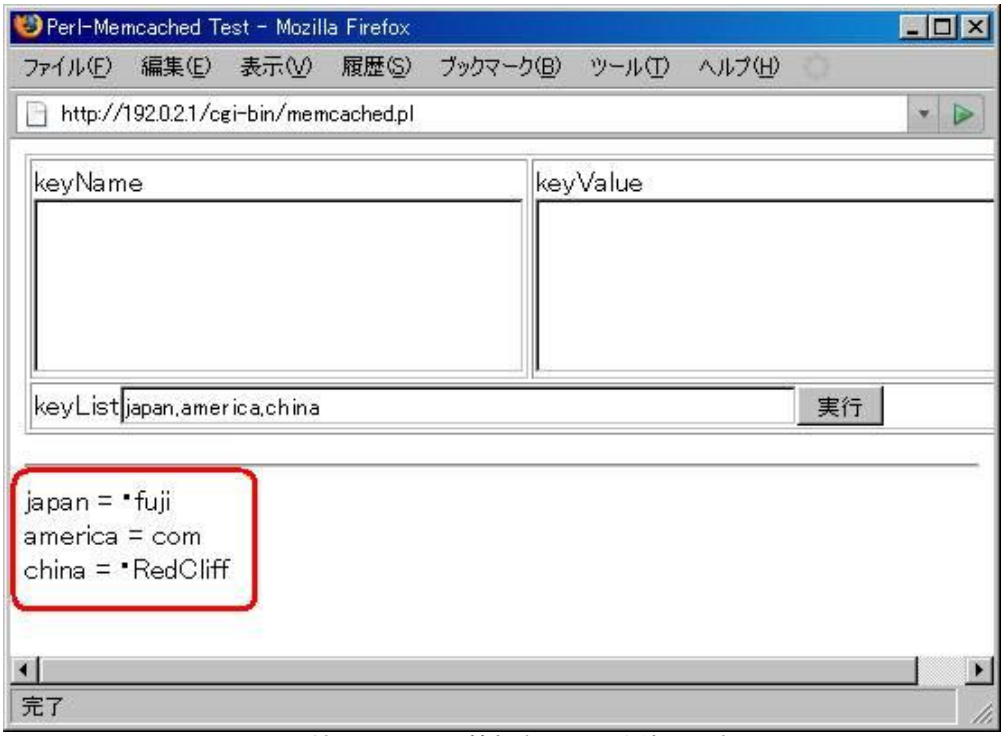

図 4.5-6 : 図 4.5-4 の結果 2。Perl の情報表示 CGI を使って表示してみると、

Command Injection が成功していることが確認できる

図 4.5-7 : 図 4.5-4 の結果 3。memcached のコンソールからも

Command Injection が成功していることが確認できる

5. Memcahced Injection の検証結果(表)

章	ライブラリ名	バージョン	ライブラリを利用する言語	サニタイジング処理の有無
4.1	Cache::Memcached	1.24	perl	なし
4.2	Memcached client for java	2.0.1 (JDK5 Compiled)	Java	URL エンコード
4.3	libmemcached	0.23	C/C++	なし
4.4	enyim.com Memcached Client	1.2.0.2	.NET Framework 2.0	エラーとして通信しない
4.5	memcacheddotnet_clientlib	1.1.5	.NET Framework 2.0	なし

6. Memcahced Injection の対策

セキュア・プログラミングの基本にエスケープ処理というものがあるが、memcached プロトコル(9 参考③)から、キー名に関して、エスケープ法に関する記述を見つけることはできなかった。
つまり、エスケープ法は規定されていないため、利用するライブラリによってはアプリケーションプログラム側で何らかしらのサニタイズ処理(バリデートまたはエンコード処理)を実施する必要がある。

memcached プロトコルによれば、キー名は半角英数字であれば良いため、何かしらの処理をアプリケーション側で施すことによって、半角英数字だけの文字集合にする、またはそれら以外の文字が含まれることを禁止すればよい。

キー名に対して半角英数字以外を禁止するような入力値検証は、Memcached Injection のバリデート処理として有効である。

また、Base16 エンコード、Base64 エンコードや URL エンコードなども、Memcached Injection の対策として有効である(9 参考⑪)。
一方で「4.2 Memcached client for java (Java の場合)」を鑑みると、”(半角スペースを「+」に置換する)URL エンコード処理” が memcached を利用したシステム開発の中でのデファクトスタンダードであるかもしれない。

7. 現実的な攻撃可能性について(まとめに代えて)

以上のように、memcached にアクセスするライブラリによっては、ライブラリ内部でサニタイズ処理を実施しているものもあれば、実施していないものもある。
サニタイズ処理も「4.2 Memcached client for java (Java の場合)」のように URL エンコード処理をエスケープ処理の代用としている場合もあれば、「4.4 enyim.com Memcached Client (ASP.NET/C#の場合その 1)」のように、キー名に改行を含む通信が発生しないようにバリデート処理している場合もある。

今後、memcached を利用したシステム開発に読者諸氏が関わるようなことがある場合、かつキー名に汚染データを与える場合には、利用するライブラリがどのようなサニタイズ処理しているかどうかを確認することを推奨する。
ライブラリ側でサニタイズ処理を行っていない場合は、アプリケーション側でサニタイズ処理をおこなう必要がある(「6 Memcahced Injection の対策」)。

現実問題として、memcached のキー名に汚染データを含ませる必要があるという場面は、非常に稀であると思われる。
また、本文書は一つの memcached デーモンのみの環境で検証したが、現実的な利用形態は複数台の memcached デーモンが起動している分散環境であると想定できる。そのような場合、データのキー名は、データの検索キー(どの IP アドレス/ポートの memcached が該当するデータを保持しているか)の一部としても利用されるため、キー名の Injection が今回のように単純に悪用可能とならない可能性が高い(検索キーが適切でないため、データの取得ができないなど)。
さらに、memcached に保持するデータそのものが機密性の高いものでない場合の方が多いのではないだろうか。
以上のことから、memcached Injection 攻撃が現実的なセキュリティ侵害事件への悪用可能性は低いと思われる。
しかしながら、memcached を利用したシステムを開発する際には、本文書で検証した内容について留意することを求められるのではないだろうか。

8. 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

9. 参考

1. memcached
   http://www.danga.com/memcached/
2. 技術評論社「Web+DB PRESS vol.47」P59～
   「特集 2 [実例から学ぶ] mixi、ニコニコ動画、livedoor　memcached ベストプラクティス」
3. Memcached プロトコル
   http://code.sixapart.com/svn/memcached/trunk/server/doc/protocol.txt
4. Perl CPAN.org – Cache::Memcached (Perl で利用できるクライアントライブラリ)
   http://search.cpan.org/~bradfitz/Cache-Memcached-1.24/
5. Memcached Client for java (Java で利用できるクライアントライブラリ)
   http://whalin.com/memcached/
6. libmemcached (C で利用できるクライアントライブラリ)
   http://tangent.org/552/libmemcached.html
7. enyim.com Memcached Client (.Net Framework2.0 で利用できるクライアントライブラリ)
   http://www.codeplex.com/EnyimMemcached/

8. memcacheddotnet_clientlib (.Net Framework2.0 で利用できるクライアントライブラリ)
http://sourceforge.net/projects/memcacheddotnet/
9. Security of HTTPHeader ver1.2 (CrLf Injection)
http://rocketeer.dip.jp/secProg/HttpSecurity003.pdf
10. Security of WebAppli&Mail ver1.0 (CrLf Injection)
http://rocketeer.dip.jp/secProg/MailSecurity001.pdf
11. RFC4648 "The Base16, Base32, and Base64 Data Encodings"
http://tools.ietf.org/html/rfc4648

10. 履歴

- 2008 年 12 月 19 日 : ver1.0 最初の公開
- 2009 年 05 月 26 日 : ver1.1 Web サイト移転に伴う最新版公開 URL の変更

11. 最新版の公開 URL

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

12. 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
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

e-mail: scan@ntt.com

以 上