



Signed Tcl Scripts

First Published: July 19, 2007

Last Update: November 30, 2007

The Signed Tcl Scripts feature allows you to create a certificate to generate a digital signature and sign a Tool Command Language (Tcl) script with that digital signature. This feature also allows you to work with existing scripts and certificates. The digital signature is verified for authentication and then run with trusted access to the Tcl interpreter. If the script does not contain the digital signature, the script may run in a limited mode for untrusted scripts, or may not run at all.

Finding Feature Information in This Module

Your Cisco IOS software release may not support all of the features documented in this module. To reach links to specific feature documentation in this module and to see a list of the releases in which each feature is supported, use the [“Feature Information for Signed Tcl Scripts”](#) section on page 29.

Finding Support Information for Platforms and Cisco IOS and Catalyst OS Software Images

Use Cisco Feature Navigator to find information about platform support and Cisco IOS and Catalyst OS software image support. To access Cisco Feature Navigator, go to <http://www.cisco.com/go/cfn>. An account on Cisco.com is not required.

Contents

- [Prerequisites for Signed Tcl Scripts, page 2](#)
- [Restrictions for Signed Tcl Scripts, page 2](#)
- [Information About Signed Tcl Scripts, page 2](#)
- [How to Configure Signed Tcl Scripts, page 3](#)
- [Configuration Examples for Signed Tcl Script, page 15](#)
- [Additional References, page 19](#)
- [Command Reference, page 20](#)
- [Feature Information for Signed Tcl Scripts, page 29](#)
- [Glossary, page 30](#)



Americas Headquarters:

Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA

© 2007 Cisco Systems, Inc. All rights reserved.

Prerequisites for Signed Tcl Scripts

For this feature to work, the Cisco IOS public key infrastructure (PKI) configuration trustpoint commands must be enabled.

For further details, see the [“Related Documents” section on page 19](#).

Restrictions for Signed Tcl Scripts

For this feature to work, you must be running the following:

- Cisco IOS Crypto image
- OpenSSL Version 0.9.7a or above
- Expect

Information About Signed Tcl Scripts

The Signed Tcl Scripts feature introduces security for the Tcl scripts. This feature allows you to create a certificate to generate a digital signature and sign a Tcl script with that digital signature. This certificate examines the Tcl scripts prior to running them. The script is checked for a digital signature from Cisco. In addition, third parties may also sign a script with a digital signature. You may wish to sign your own internally developed Tcl scripts or you could use a script developed by a third party. If the script contains the correct digital signature, it is believed to be authentic and runs with full access to the Tcl interpreter. If the script does not contain the digital signature, the script may be run in a limited mode, known as Safe Tcl mode, or may not run at all.

To create and use signed Tcl scripts, you should understand the following concepts:

- [Cisco IOS PKI, page 2](#)
- [RSA Key Pair, page 3](#)
- [Certificate and Trustpoint, page 3](#)

Cisco IOS PKI

Cisco IOS PKI provides certificate management to support security protocols such as IP security (IPsec), secure shell (SSH), and secure socket layer (SSL). A PKI is composed of the following entities:

- Peers communicating on a secure network
- At least one certification authority (CA) that grants and maintains certificates
- Digital certificates, which contain information such as the certificate validity period, peer identity information, encryption keys that are used for secure communication, and the signature of the issuing CA
- An optional registration authority (RA) to offload the CA by processing enrollment requests
- A distribution mechanism (such as Lightweight Directory Access Protocol [LDAP] or HTTP) for certificate revocation lists (CRLs)

PKI provides you with a scalable, secure mechanism for distributing, managing, and revoking encryption and identity information in a secured data network. Every routing device participating in the secured communication is enrolled in the PKI in a process where the routing device generates a Rivest, Shamir, and Adelman (RSA) key pair (one private key and one public key) and has its identity validated by a trusted routing device (also known as a CA or trustpoint).

After each routing device enrolls in a PKI, every peer (also known as an end host) in a PKI is granted a digital certificate that has been issued by a CA. When peers must negotiate a secured communication session, they exchange digital certificates. Based on the information in the certificate, a peer can validate the identity of another peer and establish an encrypted session with the public keys contained in the certificate.

RSA Key Pair

An RSA key pair consists of a public key and a private key. When setting up your PKI, you must include the public key in the certificate enrollment request. After the certificate has been granted, the public key is included in the certificate so that peers can use it to encrypt data that is sent to the router. The private key is kept on the router and used both to decrypt the data sent by peers and to digitally sign transactions when negotiating with peers.

RSA key pairs contain a key modulus value. The modulus determines the size of the RSA key. The larger the modulus, the more secure the RSA key. However, keys with large modulus values take longer to generate, and encryption and decryption operations take longer with larger keys.

Certificate and Trustpoint

A certification authority (CA), also known as a trustpoint, manages certificate requests and issues certificates to participating network devices. These services (managing certificate requests and issuing certificates) provide centralized key management for the participating devices and are explicitly trusted by the receiver to validate identities and to create digital certificates. Before any PKI operations can begin, the CA generates its own public key pair and creates a self-signed CA certificate; thereafter, the CA can sign certificate requests and begin peer enrollment for the PKI.

You can use a CA provided by a third-party CA vendor, or you can use an internal CA, which is the Cisco IOS Certificate Server.

How to Configure Signed Tcl Scripts

This section contains the following tasks:

- [Generating a Key Pair, page 4](#) (required)
- [Generating a Certificate, page 5](#) (required)
- [Signing the Tcl Scripts, page 6](#) (required)
- [Verifying the Signature, page 7](#) (required)
- [Converting the Signature into Nonbinary Data, page 8](#) (required)
- [Configuring the Router with a Certificate, page 10](#) (required)
- [Verifying the Trustpoint, page 13](#) (optional)
- [Verifying the Signed Tcl Script, page 14](#) (optional)

- [What to Do Next, page 15](#)

Generating a Key Pair

The key pair consists of a private key and a public key. The private key is intended to be kept private, accessible only to the creator. The public key is generated from the private key and is intended to be known to the public.

To generate a key pair, use the **openssl genrsa** command and then the **openssl rsa** command.

SUMMARY STEPS

1. **openssl genrsa -out *private-key-file* *bit-length***
2. **ls -l**
3. **openssl rsa -in *private-key-file* -pubout -out *public-key-file***
4. **ls -l**

DETAILED STEPS

Step 1 **openssl genrsa -out *private-key-file* *bit-length***

This command generates a private key that is *bit-length* bits long and writes the key to the *private-key-file* file.

```
Host% openssl genrsa -out privkey.pem 2048
```

```
Generating RSA private key, 2048 bit long modulus
```

```
.....+++
```

```
.....+++
```

```
e is 65537 (0x10001)
```

Step 2 **ls -l**

This command displays detailed information about each file in the current directory, including the permissions, owners, size, and when last modified.

```
Host% ls -l
```

```
total 8
```

```
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
```

The *privkey.pem* file contains the private key generated using the **openssl genrsa** command.

Step 3 **openssl rsa -in *private-key-file* -pubout -out *public-key-file***

This command generates a public key based on the specified private key in the *private-key-file* file and writes the public key to the *public-key-file* file.

```
Host% openssl rsa -in privkey.pem -pubout -out pubkey.pem
```

```
writing RSA key
```

Step 4 **ls -l**

This command displays detailed information about each file in the current directory, including the permissions, owners, size, and when last modified.

```
Host% ls -l
```

```
total 16
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12      451 Jun 12 14:57 pubkey.pem
```

The pubkey.pem file contains the public key generated from the private key using the **openssl rsa** command.

Generating a Certificate

Perform this task to generate a certificate. To generate an X.509 certificate, use the **openssl req** command.

SUMMARY STEPS

1. **openssl req -new -x509 -key *private-key-file* -out *certificate-file* -days *number***
2. **ls -l**

DETAILED STEPS

Step 1 **openssl req -new -x509 -key *private-key-file* -out *certificate-file* -days *expiration-days***

This command creates an X.509 certificate, with full access to a private key that is stored in the *private-key-file* file, and stores the certificate in the *certificate-file* file. The certificate is configured to expire in *expiration-days* days.

To complete the command, enter the following Distinguished Name (DN) information when prompted:

- Country name
- State or province name
- Organization name
- Organizational unit name
- Common name
- Email address

At each prompt, text enclosed in square brackets indicates the default value that will be used if you do not enter a value before you press Enter.

This example shows how to create an X.509 certificate that has full access to the private key in the privkey.pem file. The certificate is written to the cert.pem file and will expire 1095 days after the creation date.

```
Host% openssl req -new -x509 -key privkey.pem -out cert.pem -days 1095
```

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value, If you enter '.', the field will be left blank.

Country Name (2 letter code) [GB]:**US**

State or Province Name (full name) [Berkshire]:**California**

Locality Name (eg, city) [Newbury]:**San Jose**

```

Organization Name (eg, company) [My Company Ltd]:Cisco Systems, Inc.
Organizational Unit Name (eg, section) []:DEPT_ACCT
Common Name (eg, your name or your server's hostname) []:Jane
Email Address []:janedoe@company.com

```

Step 2 **ls -l**

This command displays detailed information about each file in the current directory, including the permissions, owners, size, and when last modified.

```

Host% ls -l
total 24
-rw-r--r--  1 janedoe eng12      1659 Jun 12 15:01 cert.pem
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12       451 Jun 12 14:57 pubkey.pem

```

The cert.pem file contains the X.509 certificate created using the **openssl req** command.

Signing the Tcl Scripts

Perform this task to sign the Tcl scripts. You will need to sign the Tcl file and output in OpenSSL document in pkcs7 (PKCS#7) format.

To sign the Tcl file, use the **openssl smime** command with the **-sign** keyword.

SUMMARY STEPS

1. **openssl smime -sign -in tcl-file -out signed-message-file -signer certificate-file -inkey private-key-file -outform DER -binary**
2. **ls -l**

DETAILED STEPS

Step 1 **openssl smime -sign -in tcl-file -out signed-tcl-file -signer certificate-file -inkey private-key-file -outform DER -binary**

This command signs the Tcl filename *tcl-file* using the certificate stored in *certificate-file* and the private key stored in *private-key-file* file and then writes the signed Tcl file in DER PKCS#7 format to the *signed-tcl-file* file.

```

Host% openssl smime -sign -in hello -out hello.pk7 -signer cert.pem -inkey privkey.pem
-outform DER -binary

```

Step 2 **ls -l**

This command displays detailed information about each file in the current directory, including the permissions, owners, size, and when last modified.

```

Host% ls -l
total 40
-rw-r--r--  1 janedoe eng12      1659 Jun 12 15:01 cert.pem
-rw-r--r--  1 janedoe eng12       115 Jun 13 10:16 hello
-rw-r--r--  1 janedoe eng12      1876 Jun 13 10:16 hello.pk7
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12       451 Jun 12 14:57 pubkey.pem

```

The hello.pk7 file contains the signed Tcl file created by the **openssl smime** command from the unsigned Tcl file named hello and using the X.509 certificate in the cert.pem file.

Verifying the Signature

Perform this task to verify that the signature matches the data, use the **openssl smime** command with the **-verify** keyword. The original Tcl content must be provided in the input file, because the file does not have the original content.

SUMMARY STEPS

1. **openssl smime -verify -in *signed-tcl-file* -CAfile *certificate-file* -inform DER -content *tcl-file***
2. **ls -l**

DETAILED STEPS

Step 1 **openssl smime -verify -in *signed-tcl-file* -CAfile *certificate-file* -inform DER -content *tcl-file***

This command verifies the signed Tcl file stored in DER PKCS#7 format in *signed-tcl-file* using the trusted Certificate Authority (CA) certificates in *certificate-file* and then writes the detached content to the file *tcl-file*.

The following example shows how to verify the signature with the input file hello.pk7:

```
Host% openssl smime -verify -in hello.pk7 -CAfile cert.pem -inform DER -content hello

puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"
Verification successful
```



Note

The SSL command page describes **-in *filename*** as the input message to be encrypted or signed or the MIME message to be decrypted or verified. For more information, go to <http://www.openssl.org/>.

Step 2 **ls -l**

This command displays detailed information about each file in the current directory, including the permissions, owners, size, and when last modified.

```
Host% ls -l
total 40
-rw-r--r--  1 janedoe eng12      1659 Jun 13 10:18 cert.pem
-rw-r--r--  1 janedoe eng12       115 Jun 13 10:17 hello
-rw-r--r--  1 janedoe eng12     1876 Jun 13 10:16 hello.pk7
-rw-r--r--  1 janedoe eng12     1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12       451 Jun 12 14:57 pubkey.pem
```

The hello file contains the content detached from the signed Tcl file hello.pk7 by running the **openssl smime** command with the **-verify** keyword. If the verification was successful, the signer's certificates are written to the X.509 certificate in the cert.pem file.

Converting the Signature into Nonbinary Data

Perform this task to convert the signature from binary to nonbinary data.

SUMMARY STEPS

1. **xxd -ps signed-tcl-file > nonbinary-signature-file**
2. Create a script that will display **#Cisco Tcl Signature V1.0** then insert the comment character **#** at the beginning of each line of the input file.
3. Run the script, supplying the name of the file containing the nonbinary signature file (*nonbinary-signature-file* file) as the input argument.
4. **ls -l**
5. **cat signed-tcl-file commented-nonbinary-signature-file > signed-tcl-script**
6. **cat signed-tcl-script**

DETAILED STEPS

Step 1 **xxd -ps signed-tcl-file > nonbinary-signature-file**

This command converts the signature in *signed-tcl-file* from binary to nonbinary data and stores it as a hexadecimal dump in the file *nonbinary-signature-file*.

```
Host% xxd -ps hello.pk7 > hello.hex
```

Step 2 Create a script that displays **#Cisco Tcl Signature V1.0** in the first line and inserts a comment character (**#**) at the beginning of each line of the input file and writes each line to a file whose name is formed by appending the text string **“_sig”** to the name of the input file.

In this example the **cat** command is used to display the contents of the script file named *my_append*.

```
Host% cat my_append
```

```
#!/usr/bin/env expect
```

```
set my_first {#Cisco Tcl Signature V1.0}
set newline {}
set my_file [lindex $argv 0]
set my_new_file ${my_file}_sig
set my_new_handle [open $my_new_file w]
set my_handle [open $my_file r]

puts $my_new_handle $newline
puts $my_new_handle $my_first
foreach line [split [read $my_handle] "\n"] {
    set new_line {#}
    append new_line $line
    puts $my_new_handle $new_line
}
```



```
close $my_new_handle
close $my_handle
```

- Step 3** Run the script, supplying the name of the file containing the nonbinary signature file (*nonbinary-signature-file*) as the input argument.

In this example, the `my_append` script is run with the nonbinary signature file `hello.hex` specified as input. The output file will be named `hello.hex_sig`.

```
Host% my_append hello.hex
```

- Step 4** `ls -l`

This command displays detailed information about each file in the current directory, including the permissions, owners, size, and when last modified.

```
Host% ls -l

total 80
-rw-r--r--  1 janedoe eng12      1659 Jun 13 10:18 cert.pem
-rw-r--r--  1 janedoe eng12       115 Jun 13 10:17 hello
-rw-r--r--  1 janedoe eng12     3815 Jun 13 10:20 hello.hex
-rw-r--r--  1 janedoe eng12     3907 Jun 13 10:22 hello.hex_sig
-rw-r--r--  1 janedoe eng12     1876 Jun 13 10:16 hello.pk7
-rwxr--r--  1 janedoe eng12      444 Jun 13 10:22 my_append
-rw-r--r--  1 janedoe eng12     1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12      451 Jun 12 14:57 pubkey.pem
```

The `hello.hex` file contains nonbinary data (stored as a hexadecimal dump) converted from the binary signature in the signed Tcl file `hello.pk7`. The `my_append` file contains the script that inserts a comment character at the beginning of each line of the input file. The `hello.hex_sig` file is the file created by running the `my_append` script on the nonbinary signature file.

- Step 5** `cat signed-tcl-file commented-nonbinary-signature-file > signed-tcl-script`

This command appends the contents of the nonbinary signature file (*commented-nonbinary-signature-file*) to the signed Tcl file stored in DER PKCS#7 format (in the *signed-tcl-file* file). The concatenated output is written to the file *signed-tcl-script*.

```
Host% cat hello hello.hex_sig > hello.tcl
```

- Step 6** `cat signed-tcl-script`

This command displays the contents of the file *signed-tcl-script*, which is the concatenation of content detached from the signed Tcl file and the nonbinary signature file.

```
Host% cat hello.tcl

puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"

#Cisco Tcl Signature V1.0
#3082075006092a864886f70d010702a08207413082073d020101310b3009
#06052b0e03021a0500300b06092a864886f70d010701a08204a13082049d
#30820385a003020102020100300d06092a864886f70d0101040500308195
#310b300906035504061302553311330110603550408130a43616c69666f
#726e69613111300f0603550407130853616e204a6f7365311c301a060355
#040a1313436973636f2053797374656d732c20496e632e310e300c060355
#040b13054e53535447310d300b060355040313044a6f686e3121301f0609
#2a864886f70d01090116126a6c6175746d616e40636973636f2e636f6d30
#1e170d3037303631323232303134335a170d313030363131323230313433
#5a308195310b300906035504061302553311330110603550408130a4361
```

```
#6c69666f726e69613111300f0603550407130853616e204a6f7365311c30
#1a060355040a1313436973636f2053797374656d732c20496e632e310e30
#0c060355040b13054e53535447310d300b060355040313044a6f686e3121
#301f06092a864886f70d01090116126a6c6175746d616e40636973636f2e
#636f6d30820122300d06092a864886f70d01010105000382010f00308201
#0a0282010100a751eb5ec1f3009738c88a55987c07b759c36f3386342283
#67ea20a89d9483ae85e0c63eeded8ab3eb7a08006689f09136f172183665
#c971099ba54e77ab47706069bbefaaab8c50184396350e4cc870c4c3f477
#88c55c52e2cf411f05b59f0eaec0678ff5cc238fdce2263a9fc6b6c244b8
#ffaead865c19c3d3172674a13b24c8f2c01dd8b1bd491c13e84e29171b85
#f28155d81ac8c69bb25ca23c2921d85fbf745c106e7aff93c72316cbc654
#4a34ea88174a8ba7777fa60662974e1fbac85a0f0aeac925dba6e5e850b8
#7caffce2fe8bb04b61b62f532b5893c081522d538005df81670b931b0ad0
#e1e76ae648f598a9442d5d0976e67c8d55889299147d0203010001a381f5
#3081f2301d0603551d0e04160414bc34132be952ff8b9e1af3b93140a255
#e54a667c3081c20603551d230481ba3081b78014bc34132be952ff8b9e1a
#f3b93140a255e54a667ca1819ba48198308195310b300906035504061302
#5553311330110603550408130a43616c69666f726e69613111300f060355
#0407130853616e204a6f7365311c301a060355040a1313436973636f2053
#797374656d732c20496e632e310e300c060355040b13054e53535447310d
#300b060355040313044a6f686e3121301f06092a864886f70d0109011612
#6a6c6175746d616e40636973636f2e636f6d820100300c0603551d130405
#30030101ff300d06092a864886f70d010104050003820101000c83c1b074
#6720929c9514af6d5df96f0a95639f047c40a607c83d8362507c58fa7f84
#aa699ec5e5bef61b2308297a0662c653ff446acfbfb6f5cb2dd162d939338
#a5e4d78a5c45021e5d4dbabb8784efbf50cab0f5125d164487b31f5cf933
#a9f68f82cd111cbab1739d7f372ec460a7946882874b0a0f22dd53acbd62
#a944a15e52e54a24341b3b8a820f23a5bc7ea7b2278bb56838b8a4051926
#af9c167274ff8449003a4e012bcf4f4b3e280f85209249a390d14df47435
#35efabce720ea3d56803a84a2163db4478ae19d7d987ef6971c8312e280a
#aac0217d4fe620c6582a48faa8ea5e3726a99012e1d55f8d61b066381f77
#4158d144a43fb536c77d6a318202773082027302010130819b308195310b
#3009060355040613025553311330110603550408130a43616c69666f726e
#69613111300f0603550407130853616e204a6f7365311c301a060355040a
#1313436973636f2053797374656d732c20496e632e310e300c060355040b
#13054e53535447310d300b060355040313044a6f686e3121301f06092a86
#4886f70d01090116126a6c6175746d616e40636973636f2e636f6d020100
#300906052b0e03021a0500a081b1301806092a864886f70d010903310b06
#092a864886f70d010701301c06092a864886f70d010905310f170d303730
#3631333137313634385a302306092a864886f70d01090431160414372cb3
#72dc607990577fd0426104a42ee4158d2b305206092a864886f70d01090f
#31453043300a06082a864886f70d0307300e06082a864886f70d03020202
#0080300d06082a864886f70d0302020140300706052b0e030207300d0608
#2a864886f70d0302020128300d06092a864886f70d010101050004820100
#72db6898742f449b26d3ac18f43a1e7178834fb05ad13951bf042e127eea
#944b72b96f3b8ecf7eb52f3d0e383bf63651750223efe69eae04287c9dae
#b1f31209444108b31d34e46654c6c3cc10b5baba887825c224ec6f376d49
#00ff7ab2d9f88402dab9a2c2ab6aa3ecceaf5a594bdc7d3a822c55e7daa
#aa0c2b067e06967f22a20e406fe21d9013ecc6bd9cd6d402c2749f8bea61
#9f8f87acfbc9e10d6ce91502e34629adca6ee855419afafe6a823333e14
#ad4c107901d1f2bca4d7ffaadddbc54192a25da662f8b8509782c76977b8
#94879453fbb00486ccc55f88db50fcc149bae066916b350089cde51a6483
#2ec14019611720fc5bbe2400f24225fc
```

Configuring the Router with a Certificate

Perform this task to configure the router with a certificate.

Prerequisites

You must already have a Cisco IOS Crypto image; otherwise you cannot configure a certificate.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **crypto pki trustpoint** *name*
4. **enrollment terminal**
5. **exit**
6. **crypto pki authenticate** *name*
7. Enter the base-encoded CA certificate.
8. **scripting tcl secure-mode**
9. **scripting tcl trustpoint name** *name*
10. (Optional) **scripting tcl trustpoint untrusted** {**execute**|**safe-execute** | **terminate**}
11. **exit**
12. (Optional) **tclsafe**

DETAILED STEPS

-
- | | |
|---------------|---|
| Step 1 | enable
Enables privileged EXEC mode. Enter your password if prompted.

Router> enable |
| Step 2 | configure terminal
Enters global configuration mode.

Router# configure terminal |
| Step 3 | crypto pki trustpoint <i>name</i>
Declares the router is to use the Certificate Authority (CA) <i>mytrust</i> and enters ca-trustpoint configuration mode.

Router(config)# crypto pki trustpoint mytrust |
| Step 4 | enrollment terminal
Specifies manual cut-and-paste certificate enrollment. When this command is enabled, the router displays the certificate request on the console terminal, allowing you to enter the issued certificate on the terminal.

Router(ca-trustpoint)# enrollment terminal |
| Step 5 | exit
Exits ca-trustpoint configuration mode and returns to global configuration mode.

Router(ca-trustpoint)# exit |
| Step 6 | crypto pki authenticate <i>name</i> |

Note Because the CA signs its own certificate, you should manually authenticate the public key of the CA by contacting the CA administrator when you perform this command.

**Note**

Use of this keyword is usually not recommended because the signature verification is not at all performed.

The **execute** keyword is provided for internal testing purposes and to provide flexibility. For example, in a situation where a certificate has expired but the other configurations are valid and you want to work with the existing configuration, then you can use the execute keyword to work around the expired certificate.

- **safe-execute**—Allows the script to run in safe mode. You can use the tclsafe command and also enter the interactive Tcl shell safe mode to explore the safe mode Tcl commands that are available. In order to get a better understanding of what is available in this limited safe mode, use the tclsafe Exec command to explore the options.
- **terminate**—Stops any script from running and reverts to default behavior. The default policy is to terminate. When the last trustpoint name is removed, the untrusted action is also removed. The untrusted action cannot be entered until at least one trustpoint name is configured for Tcl.

The following example shows how to execute the Tcl script in safe mode using the **safe-execute** keyword when the signature verification fails.

```
Router(config)# scripting tcl trustpoint untrusted safe-execute
```

Step 11 **exit**

Exits global configuration mode and returns to privileged EXEC mode.

```
Router(config)# exit
```

Step 12 **tclsafe**

(Optional) Enables the interactive Tcl shell untrusted safe mode. This allows you to manually run Tcl commands from the Cisco IOS command-line interface (CLI) in untrusted safe mode.

```
Router# tclsafe
```

Verifying the Trustpoint

To display the trustpoints that are configured in the router, use the **show crypto pki trustpoints** command.

SUMMARY STEPS

1. **enable**
2. **show crypto pki trustpoints**

DETAILED STEPS

Step 1 **enable**

This command enables privileged EXEC mode.

```
Router> enable
```

Step 2 **show crypto pki trustpoints**

This command displays the trustpoints that are configured in the router.

```
Router# show crypto pki trustpoints
```

```
Trustpoint mytrust:
  Subject Name:
    ea=janedoe@cisco.com
    cn=Jane
    ou=DEPT_ACCT
    o=Cisco
    l=San Jose
    st=California
    c=US
    Serial Number: 00
  Certificate configured.
```

Verifying the Signed Tcl Script

To verify that the Signed Tcl Script is properly running, use the **debug crypto pki transactions** command and the **tclsh flash** command.

SUMMARY STEPS

1. **enable**
2. **debug crypto pki transactions**
3. **tclsh flash:*signed-tcl-file***

DETAILED STEPS

Step 1 **enable**

This command enables privileged EXEC mode.

```
Router> enable
```

Step 2 **debug crypto pki transactions**

This command display debugging messages for the trace of interaction (message type) between the CA and the router.

```
Router# debug crypto pki transactions
```

```
Crypto PKI Trans debugging is on
```

Step 3 **tclsh flash:*signed-tcl-file***

This command executes the Tcl script in Tcl shell.



Note

The file should be a signed Tcl file.

```
Router# tclsh flash:hello.tcl
```

```
hello
argc = 0
argv =
argv0 = flash:hello.tcl
```

```
tcl_interactive = 0

Router#
*Apr 21 04:46:18.563: CRYPTO_PKI: locked trustpoint mytrust, refcount is 1
*Apr 21 04:46:18.563: The PKCS #7 message has 0 verified signers.
*Apr 21 04:46:18.563: CRYPTO_PKI: Success on PKCS7 verify!
*Apr 21 04:46:18.563: CRYPTO_PKI: unlocked trustpoint mytrust, refcount is 0
```

What to Do Next

- To get an overview of Crypto, go to “Part 5: Implementing and Managing a PKI” section of the [Cisco IOS Security Configuration Guide](#), Release 12.4T.

Configuration Examples for Signed Tcl Script

This section provided the following configuration examples:

- [Generating a Key Pair: Example, page 15](#)
- [Generating a Certificate: Example, page 15](#)
- [Signing the Tcl Scripts: Example, page 16](#)
- [Verifying the Signature: Example, page 16](#)
- [Converting the Signature with Nonbinary Data: Example, page 16](#)
- [Configuring the Router with a Certificate: Example, page 18](#)

Generating a Key Pair: Example

Generate a Private Key: Example

```
Host% openssl genrsa -out privkey.pem 2048
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Host% ls -l
total 8
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
Host%
```

Generate a Public Key from the Private Key

```
Host% openssl rsa -in privkey.pem -pubout -out pubkey.pem
writing RSA key
Host% ls -l
total 16
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12      451 Jun 12 14:57 pubkey.pem
```

Generating a Certificate: Example

```

Host% openssl req -new -x509 -key privkey.pem -out cert.pem -days 1095
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value, If you enter '.', the field will be left
blank.
-----
Country Name (2 letter code) [GB]:US
State or Province Name (full name) [Berkshire]:California
Locality Name (eg, city) [Newbury]:San Jose
Organization Name (eg, company) [My Company Ltd]:Cisco Systems, Inc.
Organizational Unit Name (eg, section) []:DEPT_ACCT
Common Name (eg, your name or your server's hostname) []:Jane
Email Address []:janedoe@company.com
Host% ls -l
total 24
-rw-r--r--  1 janedoe eng12      1659 Jun 12 15:01 cert.pem
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12       451 Jun 12 14:57 pubkey.pem

```

Signing the Tcl Scripts: Example

```

Host% openssl smime -sign -in hello -out hello.pk7 -signer cert.pem -inkey privkey.pem
-outform DER -binary
Host% ls -l
total 40
-rw-r--r--  1 janedoe eng12      1659 Jun 12 15:01 cert.pem
-rw-r--r--  1 janedoe eng12       115 Jun 13 10:16 hello
-rw-r--r--  1 janedoe eng12      1876 Jun 13 10:16 hello.pk7
-rw-r--r--  1 janedoe eng12      1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12       451 Jun 12 14:57 pubkey.pem

```

Verifying the Signature: Example

```

Host% openssl smime -verify -in hello.pk7 -CAfile cert.pem -inform DER -content hello
puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"
Verification successful

```

Converting the Signature with Nonbinary Data: Example

```

#Cisco Tcl Signature V1.0
Then append the signature file to the end of the file.
Host% xxd -ps hello.pk7 > hello.hex
Host% cat my_append
#!/usr/bin/env expect

set my_first {#Cisco Tcl Signature V1.0}
set newline {}
set my_file [lindex $argv 0]
set my_new_file ${my_file}_sig
set my_new_handle [open $my_new_file w]
set my_handle [open $my_file r]

```



```

puts $my_new_handle $newline
puts $my_new_handle $my_first
foreach line [split [read $my_handle] "\n"] {
    set new_line {#}
    append new_line $line
    puts $my_new_handle $new_line
}

close $my_new_handle
close $my_handle

Host% my_append hello.hex
Host% ls -l
total 80
-rw-r--r--  1 janedoe eng12      1659 Jun 12 15:01 cert.pem
-rw-r--r--  1 janedoe eng12      115 Jun 13 10:16 hello
-rw-r--r--  1 janedoe eng12     3815 Jun 13 10:20 hello.hex
-rw-r--r--  1 janedoe eng12     3907 Jun 13 10:22 hello.hex_sig
-rw-r--r--  1 janedoe eng12     1876 Jun 13 10:16 hello.pk7
-rwxr--r--  1 janedoe eng12      444 Jun 13 10:22 my_append
-rw-r--r--  1 janedoe eng12     1679 Jun 12 14:55 privkey.pem
-rw-r--r--  1 janedoe eng12      451 Jun 12 14:57 pubkey.pem
Host% cat hello hello.hex_sig > hello.tcl
Host% cat hello.tcl
puts hello
puts "argc = $argc"
puts "argv = $argv"
puts "argv0 = $argv0"
puts "tcl_interactive = $tcl_interactive"

#Cisco Tcl Signature V1.0
#3082075006092a864886f70d010702a08207413082073d020101310b3009
#06052b0e03021a0500300b06092a864886f70d010701a08204a13082049d
#30820385a003020102020100300d06092a864886f70d0101040500308195
#310b3009060355040613025553311330110603550408130a43616c69666f
#726e69613111300f0603550407130853616e204a6f7365311c301a060355
#040a1313436973636f2053797374656d732c20496e632e310e300c060355
#040b13054e53535447310d300b060355040313044a6f686e3121301f0609
#2a864886f70d01090116126a6c6175746d616e40636973636f2e636f6d30
#1e170d3037303631323232303134335a170d313030363131323230313433
#5a308195310b3009060355040613025553311330110603550408130a4361
#6c69666f726e69613111300f0603550407130853616e204a6f7365311c30
#1a060355040a1313436973636f2053797374656d732c20496e632e310e30
#0c060355040b13054e53535447310d300b060355040313044a6f686e3121
#301f06092a864886f70d01090116126a6c6175746d616e40636973636f2e
#636f6d30820122300d06092a864886f70d010105000382010f00308201
#0a0282010100a751eb5ec1f3009738c88a55987c07b759c36f3386342283
#67ea20a89d9483ae85e0c63eeded8ab3eb7a08006689f09136f172183665
#c971099ba54e77ab47706069bbefaaab8c50184396350e4cc870c4c3f477
#88c55c52e2cf411f05b59f0eaec0678ff5cc238fdce2263a9fc6b6c244b8
#ffaead865c19c3d3172674a13b24c8f2c01dd8b1bd491c13e84e29171b85
#f28155d81ac8c69bb25ca23c2921d85fbf745c106e7aff93c72316cbc654
#4a34ea88174a8ba7777fa06662974e1fbac85a0f0aeac925dba6e5e850b8
#7caffce2fe8bb04b61b62f532b5893c081522d538005df81670b931b0ad0
#e1e76ae648f598a9442d5d0976e67c8d55889299147d0203010001a381f5
#3081f2301d0603551d0e04160414bc34132be952ff8b9e1af3b93140a255
#e54a667c3081c20603551d230481ba3081b78014bc34132be952ff8b9e1a
#f3b93140a255e54a667ca1819ba48198308195310b300906035504061302
#5553311330110603550408130a43616c69666f726e69613111300f060355
#0407130853616e204a6f7365311c301a060355040a1313436973636f2053
#797374656d732c20496e632e310e300c060355040b13054e53535447310d
#300b060355040313044a6f686e3121301f06092a864886f70d0109011612

```

```
#6a6c6175746d616e40636973636f2e636f6d820100300c0603551d130405
#30030101ff300d06092a864886f70d010104050003820101000c83c1b074
#6720929c9514af6d5df96f0a95639f047c40a607c83d8362507c58fa7f84
#aa699ec5e5bef61b2308297a0662c653ff446acfb6f5cb2dd162d939338
#a5e4d78a5c45021e5d4dbabb8784efbf50cab0f5125d164487b31f5cf933
#a9f68f82cd111cbab1739d7f372ec460a7946882874b0a0f22dd53acbd62
#a944a15e52e54a2431b3b8a820f23a5bc7ea7b2278bb56838b8a4051926
#af9c167274ff8449003a4e012bcf4f4b3e280f85209249a390d14df47435
#35efabce720ea3d56803a84a2163db4478ae19d7d987ef6971c8312e280a
#aac0217d4fe620c6582a48faa8ea5e3726a99012e1d55f8d61b066381f77
#4158d144a43fb536c77d6a318202773082027302010130819b308195310b
#3009060355040613025553311330110603550408130a43616c69666f726e
#69613111300f0603550407130853616e204a6f7365311c301a060355040a
#1313436973636f2053797374656d732c20496e632e310e300c060355040b
#13054e53535447310d300b060355040313044a6f686e3121301f06092a86
#4886f70d01090116126a6c6175746d616e40636973636f2e636f6d020100
#300906052b0e03021a0500a081b1301806092a864886f70d010903310b06
#092a864886f70d010701301c06092a864886f70d010905310f170d303730
#3631333713713634385a302306092a864886f70d01090431160414372cb3
#72dc607990577fd0426104a42ee4158d2b305206092a864886f70d01090f
#31453043300a06082a864886f70d0307300e06082a864886f70d03020202
#0080300d06082a864886f70d0302020140300706052b0e030207300d0608
#2a864886f70d0302020128300d06092a864886f70d010101050004820100
#72db6898742f449b26d3ac18f43a1e7178834fb05ad13951bf042e127eea
#944b72b96f3b8ecf7eb52f3d0e383bf63651750223efe69eae04287c9dae
#b1f31209444108b31d34e46654c6c3cc10b5baba887825c224ec6f376d49
#00ff7ab2d9f88402dab9a2c2ab6aa3ecceaf5a594bdc7d3a822c55e7daa
#aa0c2b067e06967f22a20e406fe21d9013ecc6bd9cd6d402c2749f8bea61
#9f8f87acfb9e10d6ce91502e34629adca6ee855419afafe6a8233333e14
#ad4c107901d1f2bca4d7ffaadddbc54192a25da662f8b8509782c76977b8
#94879453fbb00486ccc55f88db50fcc149bae066916b350089cde51a6483
#2ec14019611720fc5bbe2400f24225fc
```

Configuring the Router with a Certificate: Example

```
crypto pki trustpoint mytrust
  enrollment terminal
!
!
crypto pki authentication mytrust
crypto pki certificate chain mytrust
certificate ca 00
  308204B8 308203A0 A0030201 02020100 300D0609 2A864886 F70D0101 04050030
  819E310B 30090603 55040613 02555331 13301106 03550408 130A4361 6C69666F
  726E6961 3111300F 06035504 07130853 616E204A 6F736531 1C301A06 0355040A
  13134369 73636F20 53797374 656D732C 20496E63 2E310E30 0C060355 040B1305
  4E535354 47311630 14060355 0403130D 4A6F686E 204C6175 746D616E 6E312130
  1F06092A 864886F7 0D010901 16126A6C 6175746D 616E4063 6973636F 2E636F6D
  301E170D 30363131 31373137 35383031 5A170D30 39313131 36313735 3830315A
  30819E31 0B300906 03550406 13025553 31133011 06035504 08130A43 616C6966
  6F726E69 61311130 0F060355 04071308 53616E20 4A6F7365 311C301A 06035504
  0A131343 6973636F 20537973 74656D73 2C20496E 632E310E 300C0603 55040B13
  054E5353 54473116 30140603 55040313 0D4A6F68 6E204C61 75746D61 6E6E3121
  301F0609 2A864886 F70D0109 0116126A 6C617574 6D616E40 63697363 6F2E636F
  6D308201 22300D06 092A8648 86F70D01 01010500 0382010F 00308201 0A028201
  0100BC6D A933028A B31BF827 7258BB87 A1600CF0 21090F04 2080BECC 5818688B
  74D231DF F0C365C1 07D6E206 D7651FA8 C7B230A2 3B0011E4 EA2B6A4C 1F3F27FB
  9AF449D8 FA8900B8 3E567F77 5412881B AAD9525E 3EC1D3B1 EBCE8155 D74866F1
  0940F6D1 3A2613CD F6B3595E F468B315 6DDEFF07 BBC5D521 B560AF72 D6D5FDA7
  D9D9C99D 31E3B380 5DEB7039 A1A29EF9 46ED536E 4D768048 12D48C24 59B08973
  481AD75D E741CD9E BE06EA16 9B514AE3 91184A56 A0E51B7D 4465D730 1AB3C7DD
```

```

62CA1AC9 DF30C39A 41316B8E 72289113 98080354 C7297AD7 89B627F8 ED40D924
ADF48383 1B332C7F 73C58686 6279E2A4 4BF41644 3E60F131 090D3F5D 25F0C025
43CB0203 010001A3 81FE3081 FB301D06 03551D0E 04160414 F7F4E80E F6CC4772
5F278C44 6B85F8EE 8345AB99 3081CB06 03551D23 0481C330 81C08014 F7F4E80E
F6CC4772 5F278C44 6B85F8EE 8345AB99 A181A4A4 81A13081 9E310B30 09060355
04061302 55533113 30110603 55040813 0A43616C 69666F72 6E696131 11300F06
03550407 13085361 6E204A6F 7365311C 301A0603 55040A13 13436973 636F2053
79737465 6D732C20 496E632E 310E300C 06035504 0B13054E 53535447 31163014
06035504 03130D4A 6F686E20 4C617574 6D616E6E 3121301F 06092A86 4886F70D
01090116 126A6C61 75746D61 6E406369 73636F2E 636F6D82 0100300C 0603551D
13040530 030101FF 300D0609 2A864886 F70D0101 04050003 82010100 6D12CFF8
31078DF6 94FE5CF0 8F83639B 414F32D8 069D23E2 37E182BE 7C31EC14 E87AF216
61A6CCD3 37656934 4BE4157A 400E182B EC390D1A DC130A56 B8F35BFB D2234556
24152FE8 A736B670 58CC684E 750D08AE C7739907 917B7A72 3D26BEC7 9F554CF1
5E5EF499 ABA11124 55966616 AC9C52B2 B1082DEA D962CBAF E476C575 A9DDFBFA
C4AE63F6 1D5C9F76 7B4B9CA7 52CE65C9 E65C04FC 4B7642D6 0D1A8AF4 38194B7A
CA307EC9 51DCB847 8B8C27FB 98ACEE60 0B80DC3F 36E4E252 BD731F5F 0E781E26
C1CA4120 9B0B689B BA654250 97B22A76 CC126B77 C7779AAA D3F93C3F DCF46006
2B7F7F8C 150AF889 BBEC62F1 E53B4F3B A3626CD6 05B8AB3D F8A6A361
quit
archive
log config
scripting tcl trustpoint name mytrust
scripting tcl secure-mode
!
!
end

```

Additional References

The following sections provide references related to the Signed Tcl Scripts feature.

Related Documents

Related Topic	Document Title
Cisco IOS PKI Overview: Understanding and Planning a PKI	Cisco IOS Security Configuration Guide, Release 12.4
Implementing and Managing a PKI	
PKI commands: complete command syntax, command mode, command history, defaults, usage guidelines, and examples.	Cisco IOS Security Command Reference, Release 12.4

Standards

Standard	Title
None	—

MIBs

MIB	MIBs Link
None	To locate and download MIBs for selected platforms, Cisco IOS releases, and feature sets, use Cisco MIB Locator found at the following URL: http://www.cisco.com/go/mibs

RFCs

RFC	Title
None	—

Technical Assistance

Description	Link
The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies. To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds. Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.	http://www.cisco.com/techsupport

Command Reference

This section documents only commands that are new.

- [scripting tcl secure-mode](#)
- [scripting tcl trustpoint name](#)
- [scripting tcl trustpoint untrusted](#)
- [tclsafe](#)

scripting tcl secure-mode

To enable signature verification of the interactive Tool Command Language (Tcl) scripts, use the **scripting tcl secure-mode** command in global configuration mode. To disable signature verification of the interactive Tcl scripts, use the **no** form of this command.

scripting tcl secure-mode

no scripting tcl secure-mode

Syntax Description	This command has no arguments or keywords.
---------------------------	--

Command Default	The signature verification of the interactive Tcl scripts is disabled.
------------------------	--

Command Modes	Global configuration (config)
----------------------	-------------------------------

Command History	Release	Modification
	12.4(15)T	This command was introduced.

Usage Guidelines	Use the scripting tcl secure-mode command to enable signature verification of all Tcl scripts run on the router. By default, the signature verification of the interactive Tcl scripts is disabled. You must enable the signature verification in order to verify whether the Tcl scripts match their digital signature. That would indicate they have not been altered since the digital signature was generated. If the script does not contain the digital signature, the script may run in a limited mode for untrusted script (that is, a script that has failed signature verification) or may not run at all. After receiving the results from the signature verification, the scripts are executed.
-------------------------	---

A Cisco IOS Crypto image software is required to enable this command and configure the Signed Tcl Scripts feature. The Crypto configuration commands enable the Cisco x.509 certificate storage. The **scripting tcl secure-mode** command can be enabled after the Crypto configuration trustpoint commands are enabled.

The **scripting tcl trustpoint name** command must be configured with the **scripting tcl secure-mode** command to verify the integrity of Tcl script signatures run on the router. Both commands must be configured to fully operate the feature; otherwise, a syslog message is generated:

```
*Jun 13 17:35:14.219: %SYS-6-SCRIPTING_TCL_INVALID_OR_MISSING_SIGNATURE: tcl signing
validation failed on script signed with trustpoint name mytrust, cannot run the signed TCL
script.
```

In addition, the **crypto pki trustpoint name** command provided should contain a certificate that matches the certificate that was originally used to generate the digital signature on the Tcl script.

Examples	The following example shows how to enable signature verification of the interactive Tcl scripts:
-----------------	--

```
Router(config)# crypto pki trustpoint mytrust
Router(ca-trustpoint)# enrolment terminal
```

```
Certificate has the following attributes:
    Fingerprint MD5: 1E327DBB 330936EB 2FB8EACB 4FD1133E
    Fingerprint SHA1: EE7FF9F4 05148842 B9D50FAC D76FDC9C E0703246
% Do you accept this certificate? [yes/no]: yes
Trustpoint CA certificate accepted.
% Certificate successfully imported

Router(config)# scripting tcl secure-mode
Router(config)# scripting tcl trustpoint name mytrust
```

Related Commands	Command	Description
	scripting tcl trustpoint name	Associates an existing configured trustpoint name with a certificate to verify Tcl scripts.

scripting tcl trustpoint name

To associate an existing configured trustpoint name with a certificate to verify Tool Command Language (Tcl) scripts, use the **scripting tcl trustpoint name** command in global configuration mode. To remove an existing configured trustpoint name, use the **no** form of this command.

scripting tcl trustpoint name *name*

no scripting tcl trustpoint name *name*

Syntax Description

<i>name</i>	Name of the configured trustpoint name associated with a certificate. Only one name can be associated with one certificate.
-------------	---

Command Default

A trustpoint name is not associated with a certificate to verify the Tcl scripts.

Command Modes

Global configuration (config)

Command History

Release	Modification
12.4(15)T	This command was introduced.

Usage Guidelines

Use the **scripting tcl trustpoint name** command to associate an existing configured trustpoint name with a certificate to verify Tcl scripts. This way, Tcl identifies which certificate is used for verifying the Tcl scripts. The name must match an existing configured trustpoint name, otherwise, the command is rejected with an error message on the console. You can enter the command multiple times and configure multiple trustpoint names. Once you enter the command, you cannot modify the trustpoint name. However, you can remove the trustpoint name using the **no** form of the command. You must individually remove each name. When the last name is removed, no signature checking is performed, and the untrusted script (that is, a script that has failed signature verification) action configured by the **scripting tcl trustpoint untrusted** command is also removed.

A Cisco IOS Crypto image software is required to enable this command and configure the Signed Tcl Scripts feature. The Crypto configuration commands enable the Cisco x.509 certificate storage. The **scripting tcl trustpoint name** command can be enabled after the Crypto configuration trustpoint commands are enabled.

The **scripting tcl secure-mode** command must be configured with the **scripting tcl trustpoint name** command to verify the integrity of Tcl script signatures run on the router. Both commands must be configured to fully operate this feature; otherwise, a syslog message is generated:

```
*Jun 13 17:53:31.659: %SYS-6-SCRIPTING_TCL_SECURE_TRUSTPOINT: scripting tcl secure-mode is enabled, however no scripting tcl trustpoint names configured, cannot verify signed TCL script.
```

In addition, the **crypto pki trustpoint** *name* command provided should contain a certificate that matches the certificate that was originally used to generate the digital signature on the Tcl script.

The following example shows how the **scripting tcl trustpoint name** command is used to associate existing trustpoint names. Different names can be used for different departments with certificates:

Command	Description
<code>scripting tcl secure-mode</code>	Enables signature verification of the interactive Tcl scripts.

scripting tcl trustpoint untrusted

To allow the interactive Tool Command Language (Tcl) scripts to run regardless of the scripts failing the signature check, use the **scripting tcl trustpoint untrusted** command in global configuration mode. To disallow the interactive Tcl scripts to run regardless of the scripts failing the signature check, use the **no** form of this command.

scripting tcl trustpoint untrusted {execute | safe-execute | terminate}

no scripting tcl trustpoint untrusted

Syntax Description	execute	Executes Tcl scripts even if the signature verification fails.
	 Caution	Use of this keyword is usually not recommended because the signature verification is not performed if the execute keyword is configured.
	safe-execute	Executes the Tcl script in safe mode if the signature verification fails.
	terminate	Does not run the Tcl script if the signature verification fails. The default keyword is terminate .

Command Default No script that fails signature verification can run; the script immediately stops.

Command Modes Global configuration (config)

Command History	Release	Modification
	12.4(15)T	This command was introduced.

Usage Guidelines Use the **scripting tcl trustpoint untrusted** command to allow the interactive Tcl scripts to run regardless of the scripts failing the signature check or in untrusted mode. The untrusted script (that is, a script that has failed signature verification) is not safe to use.



Use of the **execute** keyword is usually not recommended because the signature verification is not performed.

The **execute** keyword is provided for internal testing purposes and to provide flexibility. For example in a situation where a certificate has expired but the other configurations are valid and you want to work with the existing configuration, then you can use the **execute** keyword to work around the expired certificate.

The **safe-execute** keyword allows the script to run in safe mode. You can use the **tclsafe** command and also enter the interactive Tcl shell safe mode to explore the safe mode Tcl commands that are available. In order to get a better understanding of what is available in this limited safe mode, use the **tclsafe** Exec command to explore the options.

The **terminate** keyword stops any script from running and reverts to default behavior. The default policy is to terminate. When the last trustpoint name is removed, the untrusted action is also removed. The untrusted action cannot be entered until at least one trustpoint name is configured for Tcl.

**Note**

This command only applies to the Tcl shell; it does not impact other components that make use of Tcl. For example, Embedded Event Manager (EEM) cannot perform any signature checking.

Examples

The following example shows how to execute the Tcl script in safe mode if the signature verification fails:

```
Router(config)# scripting tcl trustpoint untrusted safe-execute
```

Related Commands

Command	Description
scripting tcl trustpoint name	Associates an existing configured trustpoint name with a certificate to verify Tcl scripts.
tclsafe	Enables the interactive Tcl shell untrusted safe mode.

tclsafe

To enable the interactive Tool Command Language (Tcl) shell untrusted safe mode, use the **tclsafe** command in privileged EXEC mode. To exit from the safe mode, use the **exit** or the **tclquit** command.

tclsafe

Syntax Description	This command has no arguments or keywords.
---------------------------	--

Command Default	The Tcl shell untrusted safe mode is disabled.
------------------------	--

Command Modes	Privileged EXEC (#)
----------------------	---------------------

Command History	Release	Modification
	12.4(15)T	This command was introduced.

Usage Guidelines	Use the tclsafe command when you want to manually run Tcl commands from the Cisco IOS command-line interface (CLI) in untrusted safe mode. When you use the tclsafe command and enter the interactive Tcl shell safe mode, you can explore the safe mode Tcl commands that are available. When a script fails the signature check for a configured trustpoint name, it is determined to be untrusted. Untrusted Tcl scripts execute in limited safe mode, if scripting tcl trustpoint untrusted safe-execute command is configured. In order to get a better understanding of what is available in this limited safe mode, use the tclsafe Exec command to explore the options.
-------------------------	--

After Tcl commands are entered they are sent to a Tcl interpreter. If the commands are recognized as valid Tcl commands, the command is executed and the result is sent to the tty. If a command is not a recognized Tcl command, it is sent to the Cisco IOS CLI parser. If the command is not a Tcl or Cisco IOS command, two error messages are displayed.

A predefined Tcl script can be created outside of Cisco IOS software, transferred to flash or disk memory, and run within Cisco IOS software. It is also possible to create a Tcl script and precompile the code before running it under Cisco IOS software. To exit from this mode, use the **exit** or the **tclquit** command to disable the use of the Tcl shell and return to privileged EXEC mode.

You can also use the **tclsafe** command with a script name such as **tclsafe disk0:hello.tcl**. The script **hello.tcl** executes immediately and allows you to exit from the untrusted safe mode and return to privileged EXEC mode.

Examples	The following example shows how to enable the Tcl shell untrusted safe mode and run info commands :
-----------------	--

```
Router# tclsafe
Router(safe)(tcl)# info commands
info commands
tell socket subst open eof glob list pid time eval lrange tcl_trace fblocked lsearch gets
case lappend proc break variable llength return linsert error catch clock info split array
if fconfigure concat join lreplace source fcopy global switch update close cd for file
```

```
append format read package set binary namespace scan seek while flush after vwait uplevel
continue hostname foreach rename fileevent regexp upvar unset encoding expr load regsub
interp history puts incr lindex lsort string
```

The following example shows how to execute the script **hello.tcl** to exit from the untrusted safe mode and return to privileged EXEC mode.

```
Router# tclsafe disk0:hello.tcl
```

Related Commands

Command	Description
scripting tcl trustpoint untrusted	Allows the interactive Tcl scripts to run regardless of the scripts failing the signature check.
tclquit	Quits Tcl shell.
tclsh	Enables the interactive Tcl shell and enters Tcl configuration mode.

Feature Information for Signed Tcl Scripts

Table 1 lists the release history for this feature.

Not all commands may be available in your Cisco IOS software release. For release information about a specific command, see the command reference document.

Use Cisco Feature Navigator to find information about platform support and software image support. Cisco Feature Navigator enables you to determine which Cisco IOS and Catalyst OS software images support a specific software release, feature set, or platform. To access Cisco Feature Navigator, go to <http://www.cisco.com/go/cfn>. An account on Cisco.com is not required.

**Note**

Table 1 lists only the Cisco IOS software release that introduced support for a given feature in a given Cisco IOS software release train. Unless noted otherwise, subsequent releases of that Cisco IOS software release train also support that feature.

Table 1 **Feature Information for Signed Tcl Scripts**

Feature Name	Releases	Feature Information
Signed Tcl Scripts	12.4(15)T	The Signed Tcl Scripts feature allows you to create a certificate to generate a digital signature and sign a Tcl script with that digital signature. The following commands were introduced by this feature: scripting tcl secure-mode , scripting tcl trustpoint name , scripting tcl trustpoint untrusted , and tclsafe .

Glossary

CA—certification authority. Service responsible for managing certificate requests and issuing certificates to participating IPsec network devices. This service provides centralized key management for the participating devices and is explicitly trusted by the receiver to validate identities and to create digital certificates.

certificates—Electronic documents that bind a user's or device's name to its public key. Certificates are commonly used to validate a digital signature.

CRL—certificate revocation list. Electronic document that contains a list of revoked certificates. The CRL is created and digitally signed by the CA that originally issued the certificates. The CRL contains dates for when the certificate was issued and when it expires. A new CRL is issued when the current CRL expires.

IPsec—IP security

peer certificate—Certificate presented by a peer, which contains the peer's public key and is signed by the trustpoint CA.

PKI—public key infrastructure. System that manages encryption keys and identity information for components of a network that participate in secured communications.

RA—registration authority. Server that acts as a proxy for the CA so that CA functions can continue when the CA is offline. Although the RA is often part of the CA server, the RA could also be an additional application, requiring an additional device to run it.

RSA keys—Public key cryptographic system developed by Ron Rivest, Adi Shamir, and Leonard Adleman. An RSA key pair (a public and a private key) is required before you can obtain a certificate for your router.

SHA1—Secure Hash Algorithm 1

SSH—secure shell

SSL—secure socket layer

Any Internet Protocol (IP) addresses used in this document are not intended to be actual addresses. Any examples, command display output, and figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses in illustrative content is unintentional and coincidental.

© 2007 Cisco Systems, Inc. All rights reserved.