HTTP Cookie Security

Use Case:

In HTTP world Cookie is of huge importance because if you have the cookie, in most cases you would not need to go through the authentication and authorization layer for App access. This also provides flexibility to end users because in no case you want to be authenticating every time. Also Cookie is used for making persistence decisions on ADC. Given the importance of Cookie value, one can choose to encrypt the Cookie going out in first response and when client sends it back in next request, then decrypt and match it. This way you ensure that while the packet is in transit and if at all sniffing is happening on the wire, your Cookie content is safe and secured.

F5 iRules:

```
when RULE_INIT {

        # Cookie name prefix
        set static::ck_pattern "BIGipServer*"

        # Log debug to /var/log/ltm? 1=yes, 0=no)
        set static::ck_debug 1

        # Cookie encryption passphrase
        # Change this to a custom string!
        set static::ck_pass "mypass1234"
}
when HTTP_REQUEST {

        if {$static::ck_debug}{log local0. "Request cookie names: [HTTP::cookie names]"}

        # Check if the cookie names in the request match our string glob pattern
        if {[set cookie_names [lsearch -all -inline [HTTP::cookie names] $static::ck_pattern]] ne ""}{

                # We have at least one match so loop through the cookie(s) by name
```

```
            if {$static::ck_debug}{log local0. "Matching cookie names: [HTTP::cookie names]"}
            foreach cookie_name $cookie_names {

                  # Decrypt the cookie value and check if the decryption failed (null return value)
                  if {[HTTP::cookie decrypt $cookie_name $static::ck_pass] eq ""}{

                        # Cookie wasn't encrypted, delete it
                        if {$static::ck_debug}{log local0. "Removing cookie as decryption failed for $cookie_name"}
                        HTTP::cookie remove $cookie_name
                  }
            }
            if {$static::ck_debug}{log local0. "Cookie header(s): [HTTP::header values Cookie]"}
      }
}
when HTTP_RESPONSE {

      if {$static::ck_debug}{log local0. "Response cookie names: [HTTP::cookie names]"}

      # Check if the cookie names in the request match our string glob pattern
      if {[set cookie_names [lsearch -all -inline [HTTP::cookie names] $static::ck_pattern]] ne ""}{

            # We have at least one match so loop through the cookie(s) by name
            if {$static::ck_debug}{log local0. "Matching cookie names: [HTTP::cookie names]"}
            foreach cookie_name $cookie_names {

                  # Encrypt the cookie value
                  HTTP::cookie encrypt $cookie_name $static::ck_pass
```

```
                }
                if {$static::ck_debug}{log local0. "Set-Cookie header(s)
: [HTTP::header values Set-Cookie]"}
        }
}
```

NetScaler Solution:

#Specify the Encryption method, custom keyvalue as a HexStream and optional iv as HexStream
#This way of encrypting/decrypting using custom keyValue will work only after 11.1(ENH0242353)
#For older versions, we can use Netscaler's legacy encrypt/decrypt PI function

add ns encryptionkey key_name -method <> -keyValue <> -iv <>

add rewrite action delete_cookie delete HTTP.REQ.COOKIE.VALUE("name_of_cookie")
add rewrite policy pol_delete_cookie
HTTP.REQ.COOKIE.VALUE("name_of_cookie").DECRYPT("key_name").EQ("") delete_cookie
Bind it to Request Side LB or Global at a higher priority

add rewrite action decrypt_cookie replace HTTP.REQ.COOKIE.VALUE("name_of_cookie")
HTTP.REQ.COOKIE.VALUE("name_of_cookie").decrypt("key_name")
add rewrite policy pol_decrypt_cookie
HTTP.REQ.COOKIE.VALUE("name_of_cookie").EQ("").NOT decrypt_cookie
Bind it to Request side LB or global

add rewrite action encrypt_cookie replace HTTP.RES.COOKIE.VALUE("name_of_cookie")
HTTP.RES.COOKIE.VALUE("name_of_cookie").encrypt("key_name")
add rewrite policy pol_encrypt_cookie
HTTP.RES.COOKIE.VALUE("name_of_cookie").EQ("").NOT encrypt_cookie
Bind it to Response side LB or global

In NetScaler one can define the encryption key for on the fly data encryption and decryption. We also have the encrypt/decrypt functions as part of the advance policy infrastructure which can be used for same purpose here. Technically you can apply the same logic to any other data token passing through NetScaler.

NetScaler can Encrypt/Decrypt only one cookie for a HTTP transaction, which is the need.