IMPLEMENTATION GUIDE

Voltage Security, Inc. & F5 Networks, Inc.

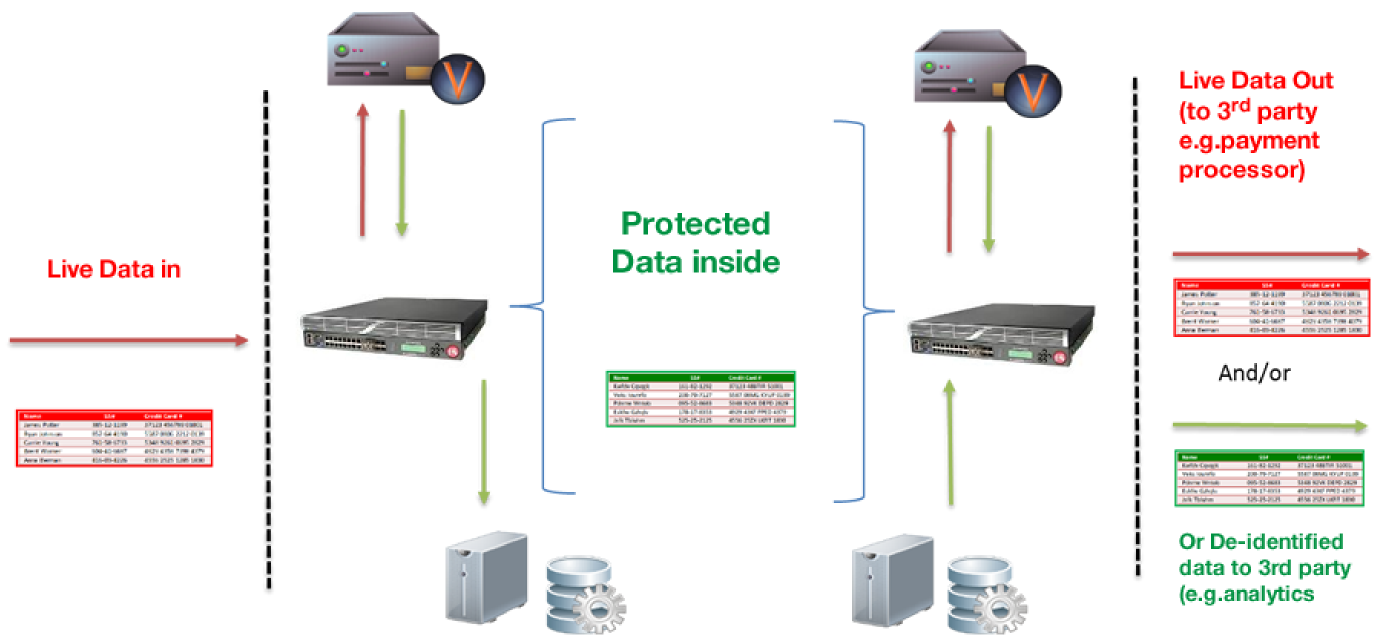
Integrated Data Protection Solution

Table of Contents

Introduction

The combination of F5 Networks® BIG-IP® Local Traffic Manager™ and Voltage SecureData™ enables data protection to take place on demand at the point sensitive data passes into a network or from application to application.

For industries subject to security audits, such as PCI DSS, this is a big benefit as data protection takes place at the edge or ingress point to the enterprise and has the effect of reducing the exposure of live and regulated data on the internal IT network. This can significantly reduce the cost and complexity of audit. Moreover, the implementation of the combined solution is relatively simple with the benefit of short time to value.


Live Data in
Protected Data inside
Live Data Out (to 3rd party e.g.payment processor)
And/or
Or De-identified data to 3rd party (e.g.analytics

Voltage SecureData and F5 BIG-IP LTM®: Real-time Protection

This Implementation Guide describes the process for integrating Voltage SecureData application programming interface (API) with F5's BIG-IP Local Traffic Manager. The F5 BIG-IP LTM traffic manager will be set with a policy to look for a specific HTML page and payload and F5 iRules® to transform data elements of the page on the fly to a de-identified and protected form.

The solution provides a data de-identification services at the network layer, while processing a Hyper Text Transport Protocol (HTTP) request. Thus the solution can be used with all application server types, including but not limited to ASP, DHTML, DO, JSP and PHP server types, without modifying the web application itself. A further benefit of encrypting sensitive data at the point of entry is this process greatly reduces the scope and cost of audits required by security regulations, such as PCI.

Product Requirements

F5 Networks BIG-IP Local Traffic Manager, version 11.0 or later.

Voltage SecureData Enterprise, version 5.0 or later.

Audience

The intended audience for this Solution Implementation Guide is system integrators needing to implement the solution. This document assumes a basic familiarity with the installation and setup of F5 BIG-IP LTM and F5 iRules.

Architecture and Process

This section describes the solution's general architecture as well as a process with detailed activities. A diagram of this process is included for reference.

Architecture Description

This solution assumes the use of F5 BIG-IP Local Traffic Manager version 11.0 or later in conjunction with the Voltage SecureData Appliance version 5.0 or later. The F5 BIG-IP LTM allows creation of virtual servers that manipulate HTTP traffic, while the Voltage SecureData Appliance provides encryption services.

Normally a system integrator creates a virtual server using the F5 BIG-IP LTM to manage traffic to and from a web application. With this solution, this first virtual server executes an F5 iRule to extract sensitive data from incoming HTTP requests.[1] We refer to this virtual server and its corresponding F5 iRule as the extractor.

Specifically for this solution, an integrator provisions a second F5 BIG-IP LTM virtual server to process encryption requests. This second virtual server runs an F5 iRule that reformats HTTP requests generated by extractor F5 iRule into SOAP requests processed by the Voltage SecureData Appliance.[2] We refer to this virtual server and its corresponding F5 iRule as the encryptor.

Process Description

The two F5 BIG-IP LTM virtual servers described above together execute a process to encrypt data at the point of entry. This process has six activities which are detailed below. See the following page for a diagram of the overall process illustrating these six activities.

1. When an end user submits a web form, the corresponding web browser generates an HTTP request.
2. The extractor, designated Web_Server_443 in the diagram, parses the incoming request. If the request contains sensitive information, such as a credit card number, the extractor creates a sideband connection to issue a HTTP GET request, passing the sensitive data as a parameter. At this time, the extractor blocks pending a response to this request.
3. The encryptor, designated Tokenization_vs in the diagram, receives this request and translates it into SOAP format, which is accepted by the Voltage SecureData Appliance. The encryptor also blocks pending a response to this SOAP request.
4. The Voltage SecureData Appliance responds with encrypted data per SOAP semantics in real-time.
5. The encryptor parses the Voltage SecureData Appliance response, generates a response to the extractor, and unblocks (continues with processing).
6. The extractor parses the response to its sideband request. Using the encrypted form of the sensitive data, the extractor modifies and sends the original user form submission to a target web server.

1 Theoretically an F5 iRule can encrypt data in either a GET or POST request, but performance increases substantially if sensitive data is limited to POST requests.

2 Theoretically it is possible to perform all of the work in one virtual server. But decoupling extraction from encryption provides more throughput.

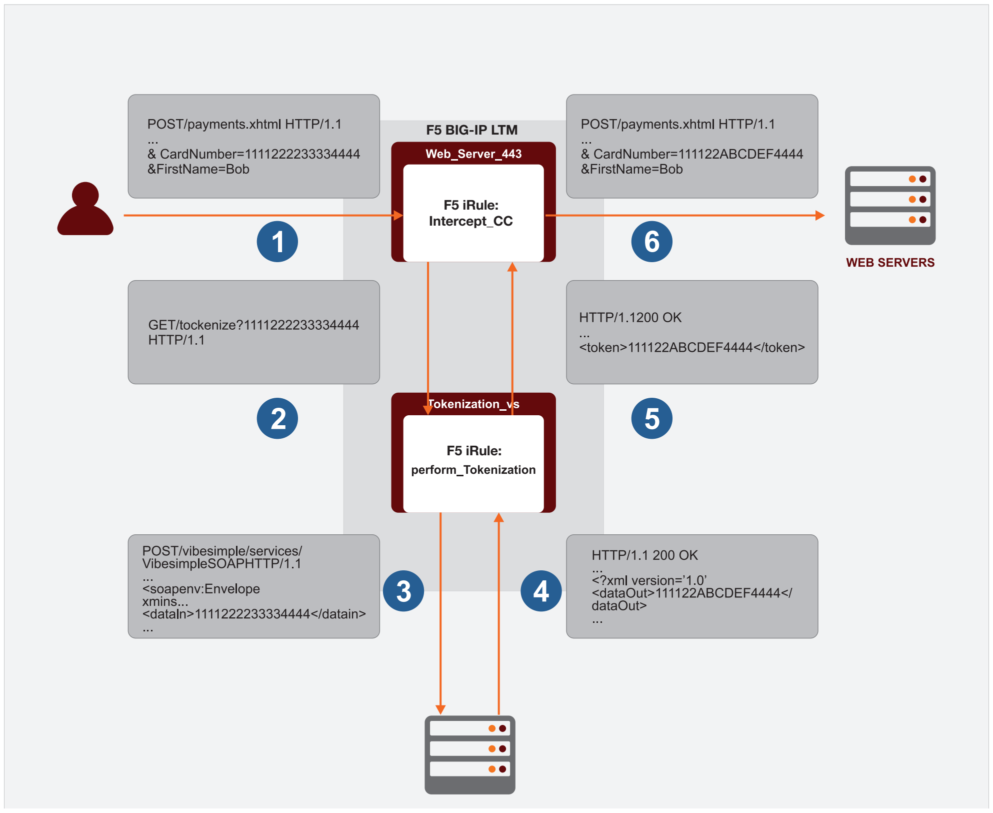

POST/payments.xhtml HTTP/1.1
...
& CardNumber=1111222233334444
&FirstName=Bob
F5 BIG-IP LTM
Web_Server_443
F5 iRule: Intercept_CC
POST/payments.xhtml HTTP/1.1
...
& CardNumber=111122ABCDEF4444
&FirstName=Bob
WEB SERVERS
1
6
GET/tockenize?1111222233334444 HTTP/1.1
HTTP/1.1200 OK
...
<token>111122ABCDEF4444</token>
2
5
Tokenization_vs
F5 iRule: perform_Tokenization
POST/vibesimple/services/ VibesimpleSOAPHTTP/1.1
...
<soapenv:Envelope
xmins...
<dataIn>1111222233334444</datain>
...
3
4
HTTP/1.1 200 OK
...
<?xml version='1.0'
<dataOut>111122ABCDEF4444</ dataOut>
...

Figure 1: Architecture & Process Diagram of the F5 BIG-IP LTM-Voltage SecureData Solution

Note the final result of this process is sensitive data contained in user's original HTTP form is substituted with encrypted values prior to application server submission. Thus the application server never processes any sensitive data. Such servers now lie outside the scope of security audits.

The following section documents the processing details of the two F5 iRules used in this solution. Note one F5 iRule runs on the extractor and the other runs on the encryptor.

Implementation Details

Since the F5 BIG-IP Local Traffic Manager (F5 BIG-IP LTM) provides resources for parsing incoming HTTP requests to manage application loading and the Voltage SecureData Appliance provides resources for sensitive data encryption, one may achieve the solution via system integration techniques. In standard F5 practice, one performs this integration by writing F5 iRules.

In its simplest form, this solution requires two F5 iRules. One runs on the F5 BIG-IP LTM virtual server that is also used for traffic management, which we refer to as the extractor, and another runs on a second F5 BIG-IP LTM virtual server specifically provisioned for data protection, which we refer to as an encryptor. This section documents the detailed design for each of these F5 iRules. Note that as a minimum, a system integrator will need to substitute our example parameters with real world values for each unique implementation.

Extractor iRule

This section describes the functions of the F5 iRule running on the extractor. We also include a coding example.

Description

The extractor F5 iRule, named Intercept_Credit Card in this example, generates and sends a GET request to the encryptor when a page contains sensitive data. After receiving a response from the encryptor, cipher text is substituted before forwarding the page to the web application server. Pages without sensitive data are passed on without modification.

The extractor F5 iRule should have as little impact as possible on any request that is not carrying any sensitive data. Accordingly, this rule immediately returns if the request is a GET, since in this example submission of a payment page including sensitive data will always be a POST request. Also, if the target of the request does not match the URL of the page collecting sensitive data, this F5 iRule returns. This processing is done in the HTTP_REQUEST event of the iRule, before any data has even been received.

The extractor F5 iRule parses data from POST requests and extracts credit card numbers from the form data. Note that this does create a dependency between the web form design and this F5 iRule. It is possible to extract the value using a regular expression, but this is less efficient and has a higher chance of returning a false positive.

Having found the credit card number, the data stream is split into two parts: everything before the first character of the number, and everything after the last character. These parts are held in temporary variables. Meanwhile, the extractor opens a sideband connection to the encryptor, and submits the credit card number to the encryptor for tokenization. Note in more complex setups, the extractor might also pass of the tokenization format and authentication information as parameters. But instead, we use constants for these parameters.

When the encryptor returns cipher text on the sideband connection, the response is parsed to extract the tokenized value. Then the extractor assembles a new payload from both parts parts of the original held in the temporary variables mentioned previously.

Coding Example

The following example F5 iRule implements processing for the extractor.

This is the rule initialization section. Variable values will vary by environment. The section executes once at virtual server startup.

```
# Set variables that will vary by environment
when RULE_INIT {
  # The target of packets that we want this rule to act on
  set static::Tokenization_URI "index.html"

  # The text that precedes the actual cc number
  set static::creditCardPrefix "cardNumber="

  # The length of this text
  set static::prefixLength 11

  # The IP address that the sideband connection should
  # connect from (the myaddr argument)
  set static::myaddr "192.168.0.100"

  # the name of the encryptor Virtual Server and to which
  # the sideband connection will be made
  set static::TokenizationVirtualServer "Tokenization_vs"

  # Value to set in the Host: header of the
  # GET request, the destination web server
  set static::HostString "ws.sd.davetest.com:443"

  # how many times we will check the connection?
  set static::retries 100

  # How long each time will have before it times out,
  # in milliseconds? The effective timeout is the product
  # of the retries and the timeout variables
  set static::timeout 10
} # end when RULE_INIT
```

The processing done when a request arrives.

GET requests are ignored…

…as are requests for any URI other than the target.

Otherwise, we collect the data in the request.

```
when HTTP_REQUEST {
  #Ignore GET requests by immediately returning from them
  #Request will then be forwarded to application server
  #without encryption
  if {[HTTP::method] equals "GET"} {
    return
  }

  #Ignore POST request from non-sensitive pages
  if {[HTTP::uri] contains $static::Tokenization_URI} {
    set Tokenize_Me 1
  } else {
    return
  }
  #Check for Content-Length header, and then collect data
  if {[HTTP::header exists "Content-Length"]} {
    HTTP::collect [HTTP::header "Content-Length"]
  } else {
    HTTP::collect 1048000
  }
} # end when HTTP_REQUEST

when HTTP_REQUEST_DATA {
  #Check whether we need to process this data
  if {[info exists Tokenize_Me]} {
    unset Tokenize_Me
  } else {
  #not aimed at the payment page
  return
} #end if info exists
#Compute position of credit card number
set pos [string first $static::creditCardPrefix[HTTP::payload]]
#Quit if not found
if {not $pos} return #nothing to do

#Compute credit card parsing parameters
set startOfActualNumberPos [expr {$pos + $static::prefixLength}]

# find the credit card number
set ccNum [substr [HTTP::payload] $startOfActualNumberPos "&"]
set FirstPartOfInboundRequest [getfield [HTTP::payload] $ccNum 1]
set SecondPartOfInboundRequest [getfield [HTTP::payload] $ccNum 2]

# Build simple GET request to encrypt credit card number
set tokenizationRequest "GET /tokenize?data=$ccNum HTTP/1.1\r\nUser-
Agent: AlmostCurl!\r\nHost: ${static::HostString}\r\nAccept: */*\r\n\
r\n"

# Connect to encryptor virtual server
set TokenServer [connect -protocol TCP -myaddr $static::myaddr -timeout
100 -idle 5 -status connect_status $static::TokenizationVirtualServer]
set Token [findstr $recv_data "<token>" 7 "</token>"]
set TokenizedData "$FirstPartOfInboundRequest$Token$SecondPartOfInbound
Request"

# Optional logging statement for debugging
#log local0. "TokenizedData : $TokenizedData"

#replace existing payload
HTTP::payload replace 0 [string length [HTTP::payload]] $TokenizedData
} #end if HTTP_REQUEST_DATA
```

Code run when data has been received.

Check whether we need to process this data…

…if so, parse out the card number from the payload…

…and store the rest of the request that surrounded the card number.

Now create the request that will be sent to the Voltage servers…

…specify where to send the request…

…and send it.

Start checking for a response from the tokenization servers.

Parse the token out of the response…

…and reconstruct the request payload, using the token

Encryptor iRule

In this section we describe the function of the F5 iRule running on the encryptor. We also present a coding example.

Description

The function of the encryptor F5 iRule, named Perform_Tokenization in this example, is to create a SOAP request, send plain text data to the Voltage SecureData Appliance, collect cipher text from the Voltage SecureData Appliance, and return the result to the extractor.

This F5 iRule performs standard text handling and parsing functions. The only Voltage-specific part of the code is the part that creates the actual SOAP message that will be sent as a POST request to the Voltage servers.

Coding Example

The following example F5 iRule implements processing for the encryptor.

```
when RULE_INIT {
  # Implementation-specific parameters
  set static::identity "dave@voltage.com"
  set static::tweak ""
  set static::authmethod "SharedSecret"
  set static::authinfo "voltage123"
  set static::format "CC_Token"
}
when CLIENT_ACCEPTED {
  TCP::collect
}
when CLIENT_DATA {
  if {[TCP::payload] contains "GET /tokenize?"} {
    # Optional logging for debugging
    #log local0. "Tokenization GET request"
  } else {
    #log local0. "Some other request - let it through"
    TCP::release
    return
}#end when CLIENT_DATA

# get the query string: this what we want to encrypt
set plaintext [findstr [TCP::payload] "data=" 5 "&"]

# Build the SOAP query
set SOAPQuery "<soapenv:Envelope xmlns:soapenv=\"http://schemas.
xmlsoap.org/soap/envelope/\" xmlns:xsd=\"http://www.w3.org/2001/
XMLSchema\" xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"
xmlns:vib=\"http://voltage.com/vibesimple\”><soapenv:Body><vib:ProtectF
ormattedData><dataIn>$plaintext</dataIn><format>$static::format</format
><identity>$static::identity</identity><tweak>$static::tweak</tweak><au
thMethod>$static::authmethod</authMethod><authInfo>$static::authinfo</
authInfo></vib:ProtectFormattedData></soapenv:Body></soapenv:Envelope>"

set uri [findstr [TCP::payload] "GET " 4 " HTTP/1."]
regsub -all -nocase "GET" [TCP::payload] "POST" newdata
#log local0. "newdata          : $newdata"
```

Set per-installation values. These could also be passed as parameters from the Intercept_CC iRule.

…and send it.

We'll need the content of the request

Check that this is the request we're expecting. Here we elect to allow other requests through, although there should never be any.

Extract the data…

…and construct the SOAP query. Here we are using the Protect Formatted Data web service method. Consult the Voltage Web Service Programmer's Guide for other options

Some text manipulation to turn this GET request into a POST before sending it to the Voltage servers.

```
# Set the target of the POST
set newdata1 [string map "$uri /vibesimple/services/VibeSimpleSOAP"
$newdata]

# Add necessary new headers
set newHeaders "Accept-Encoding: gzip,deflate\r\nContent-Type: text/
xml;charset=UTF-8\r\nSOAPAction: \"http://voltage.com/vibesimple/
ProtectFormattedData\"\r\nContent-Length: [string length $SOAPQuery]\
r\n"
regsub -all -nocase "Accept: " $newdata1 "${newHeaders}Accept: "
newdata2

set newdata3 ${newdata2}$SOAPQuery
TCP::payload replace 0 [TCP::payload length] $newdata3
TCP::release
when HTTP_RESPONSE {
  # Collect HTTP response data
  if { [HTTP::header exists "Content-Length"]}{
    HTTP::collect [HTTP::header "Content-Length"]
  } else {
  # assuming we're doing one-at-a-time (as we must
  # be, since this explicitly calls ProtectFormattedData,
  # not the List variant
  # Given that, 300 bytes is plenty
    HTTP::collect 300
  }
} #end when HTTP_RESPONSE


when HTTP_RESPONSE_DATA {
  set wholeResponse [HTTP::payload]
  set token [findstr $wholeResponse "<dataOut xmlns=\"\">" 18 "</
dataOut>"]
  set newContent "<token>$token</token>"

  HTTP::payload replace 0 [string length [HTTP::payload]]
${newContent}\r\n

  # not really worth maintaining the Chunked-Encoding
  HTTP::header remove Transfer-Encoding
  HTTP::header replace Content-Length [string length  [HTTP::payload]]

} # end when HTTP_RESPONSE_DATA
```

Set the target of the POST request: this is important

Construct the POST request in steps.

Replace the request payload and send it on

Collect the response data

Parse the token out of the response from the Voltage server, place it in an easily parsed response and send it back to the caller (the Intercept_CC F5 iRule)

References

Protection of PII, PHI, and PCI Data for Enterprises Handling Sensitive Information

http://www.voltage.com/products/securedata-enterprise/encryption/

Preserving Critical Business Functions by Maintaining Data Format

Format Preserving Encryption: **http://www.voltage.com/technology/format-preserving-encryption/**

Secure Stateless Tokenization: **http://www.voltage.com/technology/secure-stateless-tokenization/**

Voltage Security Corporate Fact Sheet

http://www.voltage.com/wp-content/uploads/Voltage_Corporate_Fact_Sheet.pdf

F5 iRule Programming Resources

F5 Configuration Guide For BIG-IP Local Traffic Management, Chapter 17, Writing iRules:

https://support.f5.com/kb/en-us/products/big-ip_ltm/manuals/product/ltm_configuration_guide_10_0_0/ltm_rules.html

F5 Dev Central, iRules Forum: **https://devcentral.f5.com/irules**

Tcl Reference Manual: **http://tmml.sourceforge.net/doc/tcl/index.html**

Voltage Security API Programing Resources

Voltage SecureData Sandbox Data Sheet: **http://www.voltage.com/products/voltage-securedata-sandbox/**

Voltage SecureData Sandbox Registration Page: **http://www.voltage.com/products/voltage-securedata-sandbox/**

Help and Support

Please contact: **F5_Solution@Voltage.com**

Support: **(866) 440-8917**