|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | Integration test:  We started a database server and a client. Then, the client sent a SQL query to the server and then received the SQL result from the server. |
| **Description** | In org.hsqldb.cmdline.SqlFile,  public String streamToString(InputStream is, final String cs)  throws IOException {  while (bytesread < ba.length &&  (retval = is.read(  ba, bytesread, ba.length - bytesread)) > 0) {  bytesread += retval;  }  }  ......  private void condlPrint(final String s, final boolean printHtml) {  if ((printHtml && !htmlMode) || (htmlMode && !printHtml)) return;  if (shared.psStd != null) shared.psStd.print(s);  if (pwQuery != null) {  pwQuery.print(s);  pwQuery.flush();  }  }    The sensitive information (String s) may be leaked.  We may be able to add control on it. |
| **Taint flow path** | //source **<org.hsqldb.cmdline.SqlFile: java.lang.String streamToString(java.io.InputStream,java.lang.String)>** -- >  <org.hsqldb.jdbc.JDBCResultSetMetaData: java.lang.String getColumnLabel(int)> -- >  ……  <org.hsqldb.lib.LineGroupReader: boolean isNewSectionLine(java.lang.String)> -- >  ……  **<org.hsqldb.cmdline.SqlFile: void condlPrint(java.lang.String,boolean)>** //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | Integration test:  We started a database server and a client. Then, the client sent a SQL query to the server and then received the SQL result from the server. |
| **Description** | In org.hsqldb.cmdline.SqlFile,  public String streamToString(InputStream is, final String cs)  throws IOException {  while (bytesread < ba.length &&  (retval = is.read(  ba, bytesread, ba.length - bytesread)) > 0) {  bytesread += retval;  }  }  ......  private void displaySqlResults(Statement statement,  ResultSet r, final int[] incCols, final String filterString,  final boolean updateStatus) throws SQLException, SqlToolError {  ......  if (excludeSysSchemas && interactive)  stdprintln(SqltoolRB.vendor\_nosup\_sysschemas.getString());  ......  for (int i = 0; i < headerArray.length; i++) {  dsvSafe(headerArray[i]);  pwDsv.print(headerArray[i]);  if (i < headerArray.length - 1)  pwDsv.print(dsvColDelim);  }  pwDsv.print(dsvRowDelim);  ......  for (int j = 0; j < fArray.length; j++) {  if (pwDsv == null) dsvSafe(fArray[j]);  pwDsv.print((fArray[j] == null)  ? (autonulls[j] ? "" : nullRepToken)  : fArray[j]);  if (j < fArray.length - 1) pwDsv.print(dsvColDelim);  }  pwDsv.print(dsvRowDelim);    The sensitive information (for headerArray and dsvColDelim) may be leaked.  We may be able to add control on them. |
| **Taint flow path** | //source **<org.hsqldb.cmdline.SqlFile: java.lang.String streamToString(java.io.InputStream,java.lang.String)>** -- >  ……  <java.sql.ResultSet: java.lang.String getString(int)> -- >  ……  **<org.hsqldb.cmdline.SqlFile: void displaySqlResults(java.sql.Statement,java.sql.ResultSet,int[],java.lang.String,boolean)>** //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | System test in the project package:  During the system testing, we started one HSQLDB server, then we executed " org.hsqldb.test.AllTests" |
| **Description** | In org.hsqldb.ParserDDL, private StatementSchema compileRoleGrantOrRevoke(boolean grant) { ...... while (true) { checkIsSimpleName(); roleList.add(token.tokenString); read();  if (token.tokenType == Tokens.COMMA) {  read();  continue;  }  break;  }  ......  }  In org.hsqldb.result.Result, public void write(SessionInterface session, DataOutputStream dataOut, RowOutputInterface rowOut) throws IOException { ...... parameterMetaData.write(rowOut); ...... } The sensitive information (rowOut) may be leaked. We may be able to add control on them. |
| **Taint flow path** | //source **<org.hsqldb.ParserDDL: org.hsqldb.StatementSchema compileRoleGrantOrRevoke(boolean)>** -- >  ......  <org.hsqldb.StatementSchema: void setOrCheckObjectName(org.hsqldb.Session,org.hsqldb.HsqlNameManager$HsqlName,org.hsqldb.HsqlNameManager$HsqlName,boolean)> -- >  ......  **<org.hsqldb.result.Result: void write(org.hsqldb.SessionInterface,org.hsqldb.lib.DataOutputStream,org.hsqldb.rowio.RowOutputInterface)>** //sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | System test in the project package:  During the system testing, we started one HSQLDB server, then we executed " org.hsqldb.test.AllTests" |
| **Description** | In org.hsqldb.dbinfo.DatabaseInformationFull{ Table TRIGGERS(Session session, PersistentStore store) { ...... row[action\_order] = ValuePool.getLong(order); ...... } }  In org.hsqldb.result.Result, public void write(SessionInterface session, DataOutputStream dataOut, RowOutputInterface rowOut) throws IOException { ...... parameterMetaData.write(rowOut); ...... } The sensitive information (rowOut) may be leaked. We may be able to add control on them. |
| **Taint flow path** | //source **<org.hsqldb.dbinfo.DatabaseInformationFull: org.hsqldb.Table TRIGGERS(org.hsqldb.Session,org.hsqldb.persist.PersistentStore)>** -- >  ......  <org.hsqldb.map.ValuePool: java.lang.Long getLong(long)> -- >  <org.hsqldb.map.ValuePoolHashMap: java.lang.Long getOrAddLong(long)> -- >  ......  **<org.hsqldb.result.Result: void write(org.hsqldb.SessionInterface,org.hsqldb.lib.DataOutputStream,org.hsqldb.rowio.RowOutputInterface)>** //sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | System test in the project package:  During the system testing, we started one HSQLDB server, then we executed " org.hsqldb.test.AllTests" |
| **Description** | In org.hsqldb.Database, synchronized Session connect(String username, String password, String zoneString, int timeZoneSeconds) { ...... if (username.equalsIgnoreCase("SA")) { username = "SA"; } ...... }  Java is case sensitive language. Strings "SA", "Sa", "sA", and "sa" are different. Thus Hsqldb should only have one super user "SA" or "sa". |
| **Taint flow path** | //source **<org.hsqldb.dbinfo.DatabaseInformationFull: org.hsqldb.Table TRIGGERS(org.hsqldb.Session,org.hsqldb.persist.PersistentStore)>** -- >  ......  <org.hsqldb.map.ValuePool: java.lang.Long getLong(long)> -- >  <org.hsqldb.map.ValuePoolHashMap: java.lang.Long getOrAddLong(long)> -- >  ...... |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | Integration test:  We started a database server and a client. Then, the client sent a SQL query to the server and then received the SQL result from the server. |
| **Description** | package org.hsqldb.cmdline;  public class SqlFile {  private static FrameworkLogger logger =  FrameworkLogger.getLog(SqlFile.class);  ......  private void displaySqlResults(Statement statement,  ResultSet r, final int[] incCols, final String filterString,  final boolean updateStatus) throws SQLException, SqlToolError {  try {  ......  } finally {  if (r != null) try {  r.close();  } catch (SQLException se) {  logger.warning("Failed to close SQL result set: " + se);  } finally {  r = null;  }  if (statement != null) try {  statement.close();  } catch (SQLException se) {  logger.warning("Failed to close SQL statement: " + se);  } finally {  statement = null;  }  }  }  Two statements logger.warning("Failed to close SQL result set: " + se) and logger.warning("Failed to close SQL statement: " + se) should be logger.error("Failed to close SQL result set: " + se) and logger.error("Failed to close SQL statement: " + se), respectively.  The reason is that they record exception messages. |
| **Taint flow path** | //source **<org.hsqldb.cmdline.SqlFile: java.lang.String streamToString(java.io.InputStream,java.lang.String)>** -- >  ……  <java.sql.ResultSet: java.lang.String getString(int)> -- >  ……  **<org.hsqldb.cmdline.SqlFile: void displaySqlResults(java.sql.Statement,java.sql.ResultSet,int[],java.lang.String,boolean)>** //Sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | HSQLDB |
| **Input** | System test in the project package:  During the system testing, we started one HSQLDB server, then we executed " org.hsqldb.test.AllTests" |
| **Description** | package org.hsqldb.lib;  public class RCData {  public Connection getConnection(String curDriverIn, String curTrustStoreIn)  throws ClassNotFoundException,  MalformedURLException,  SQLException {  ......  try {  urlString = expandSysPropVars(url);  } catch (IllegalArgumentException iae) {  throw new MalformedURLException(iae.toString() + " for URL '"  + url + "'");  }  String userString = null;  if (username != null) try {  userString = expandSysPropVars(username);  } catch (IllegalArgumentException iae) {  throw new MalformedURLException(iae.toString()  + " for user name '" + username  + "'");  }  ...  }  }  Sensitive messages (URL and username) are outputted directly and may leak when throwing MalformedURLException.  logger.error(...) may be better than 'throw new MalformedURLException(...)'. |
| **Taint flow path** | //source **<org.hsqldb.ParserBase: java.lang.String getLastPart()>** -- >  ......  <org.hsqldb.ParserDDL: org.hsqldb.StatementSchema compileCreate()> -- >  ......  **<org.hsqldb.lib.RCData: void <init>(java.io.File,java.lang.String)>** //sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Netty |
| **Input** | Integration test:  We develop a 3-tier application with three nodes. The first node read an email list from a file and then sent relevant emails to the second node. Next, the second node  encrypted the emails using RSA algorithm and then sent them to the third node. Lastly, the third node used Postfix to send emails received. |
| **Description** | Some log level checks (logger.isWarnEnabled()) should be removed.  For the short parameter which is generated simply, logger.isWarnEnabled() check is not necessary.  The problem exists in io.netty.channel.nio.AbstractNioChannel. |
| **Taint flow path** | **<io.netty.channel.nio.AbstractNioChannel: void doBeginRead()>** --> // Source  <io.netty.channel.AbstractChannel$AbstractUnsafe: void beginRead()> -->  <io.netty.channel.nio.AbstractNioChannel: void doBeginRead()> -->  <io.netty.channel.AbstractChannelHandlerContext: boolean isNotValidPromise(io.netty.channel.ChannelPromise,boolean)> -->  <io.netty.util.internal.PlatformDependent: boolean hasUnsafe()> -->  <io.netty.util.internal.ObjectUtil: java.lang.Object checkNotNull(java.lang.Object,java.lang.String)> -->  <io.netty.channel.AbstractChannel: io.netty.channel.Channel$Unsafe unsafe()> -->  ……  <io.netty.channel.AbstractChannelHandlerContext: void <init>(io.netty.channel.DefaultChannelPipeline,io.netty.util.concurrent.EventExecutor,java.lang.String,boolean,boolean)> -->  <io.netty.util.concurrent.SingleThreadEventExecutor: void <init>(io.netty.util.concurrent.EventExecutorGroup,java.util.concurrent.Executor,boolean,int,io.netty.util.concurrent.RejectedExecutionHandler)> -->  **<io.netty.channel.nio.NioEventLoop: void processSelectedKey(java.nio.channels.SelectionKey,io.netty.channel.nio.AbstractNioChannel)>** //Sink |
| **Status** | Fixed |

|  |  |
| --- | --- |
| **Subject** | Raining Sockets |
| **Input** | Integration test:  We started a server and a client, and then the client sent a text messages to the server. |
| **Description** | In raining.core.NioSocket,  static {  try {  selector = Selector.open();  } catch (Exception exc) { System.err.println( "Opening a selector L46 EXC:"+ exc.toString()); exc.printStackTrace(); }  }    Raining Sockets may leak sensitive information (of selector) through printing it directly.  Raining Sockets can use log to export information such as log.error("Opening a selector L46 EXC:", exc); |
| **Taint flow path** | //source **<raining.core.NioSocket: void poll(long)> -- >**  <raining.core.NioSocket: void start(long)> -- >  ......  **<raining.core.NioSocket: void poll(long)>** //sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | RocketMQ |
| **Input** | Integration test:  RocketMQ have four components: a name server, a broker, a producer, and a consumer. The server provides corresponding reading and writing service and records full routing information. The broker stores messages. The producer sends messages to the broker. The customer receives messages from the broker. |
| **Description** | In org.apache.rocketmq.store.ConsumeQueue,  private static final Logger log = LoggerFactory.getLogger(LoggerName.STORE\_LOGGER\_NAME);  private static final Logger LOG\_ERROR = LoggerFactory.getLogger(LoggerName.STORE\_ERROR\_LOGGER\_NAME);  ......  public void putMessagePositionInfoWrapper(DispatchRequest request) {  ......  log.error("[BUG]consume queue can not write, {} {}", this.topic, this.queueId);  this.defaultMessageStore.getRunningFlags().makeLogicsQueueError();  }  The log.error(...) should be LOG\_ERROR.error(...) because it records an error. |
| **Taint flow path** | //source **<org.apache.rocketmq.common.message.MessageDecoder: org.apache.rocketmq.common.message.MessageExt decode(java.nio.ByteBuffer,boolean,boolean,boolean)>** -- >  ......  <org.apache.rocketmq.common.protocol.header.PullMessageRequestHeader: void setQueueOffset(java.lang.Long)>-- >  ......  **<org.apache.rocketmq.store.ConsumeQueue: boolean putMessagePositionInfo(long,int,long,long)>**//sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | RocketMQ |
| **Input** | Integration test:  RocketMQ have four components: a name server, a broker, a producer, and a consumer. The server provides corresponding reading and writing service and records full routing information. The broker stores messages. The producer sends messages to the broker. The customer receives messages from the broker. |
| **Description** | package org.apache.rocketmq.namesrv;  public class NamesrvStartup {  ......  public static NamesrvController main0(String[] args) {  String file = commandLine.getOptionValue('c');  if (file != null) {  ......  System.out.printf("load config properties file OK, " + file + "%n");  ......  }  }  The System.out.printf(...) should be log.info(...) to prevent sensitive message (file) leakage. |
| **Taint flow path** | //source **<org.apache.rocketmq.store.ConsumeQueue: void recover()>** -- >  ......  <org.apache.rocketmq.store.ConsumeQueue: boolean putMessagePositionInfo(long,int,long,long)>-- >  ......  **<org.apache.rocketmq.namesrv.NamesrvStartup: org.apache.rocketmq.namesrv.NamesrvController main0(java.lang.String[])>** //sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | RocketMQ |
| **Input** | Integration test:  RocketMQ have four components: a name server, a broker, a producer, and a consumer. The server provides corresponding reading and writing service and records full routing information. The broker stores messages. The producer sends messages to the broker. The customer receives messages from the broker. |
| **Description** | package org.apache.rocketmq.remoting.netty  public class NettyEncoder extends MessageToByteEncoder<RemotingCommand> {  private static final Logger log = LoggerFactory.getLogger(RemotingHelper.ROCKETMQ\_REMOTING);  ......  public void encode(ChannelHandlerContext ctx, RemotingCommand remotingCommand, ByteBuf out)  throws Exception {  try {  ......  } catch (Exception e) {  log.error("encode exception, " + RemotingHelper.parseChannelRemoteAddr(ctx.channel()), e);  if (remotingCommand != null) {  log.error(remotingCommand.toString());  }  RemotingUtil.closeChannel(ctx.channel());  }  }  The second log.info(...) should be log.info(...) to record the information (of remotingCommand). |
| **Taint flow path** | //source **<org.apache.rocketmq.common.message.MessageDecoder: org.apache.rocketmq.common.message.MessageExt decode(java.nio.ByteBuffer,boolean,boolean,boolean)>** -- >  ......  <org.apache.rocketmq.remoting.common.RemotingHelper: java.lang.String parseChannelRemoteAddr(io.netty.channel.Channel)>-- >  ......  //sink **<org.apache.rocketmq.remoting.netty.NettyEncoder: void encode(io.netty.channel.ChannelHandlerContext,org.apache.rocketmq.remoting.protocol.RemotingCommand,io.netty.buffer.ByteBuf)>** |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | RocketMQ |
| **Input** | System test in the project package:  RocketMQ have four components: a name server, a broker, a producer, and a consumer.  During the system testing, we started a name server and a broker. Then, I executed " "org.apache.rocketmq.example.benchmark.Producer" as a producer, and executed " "org.apache.rocketmq.example.benchmark.Consumer " as a consumer. |
| **Description** | package org.apache.rocketmq.store;  public class MappedFileQueue {  private static final Logger log = LoggerFactory.getLogger(LoggerName.STORE\_LOGGER\_NAME);  private static final Logger LOG\_ERROR = LoggerFactory.getLogger(LoggerName.STORE\_ERROR\_LOGGER\_NAME);  ......  public MappedFile findMappedFileByOffset(final long offset, final boolean returnFirstOnNotFound) {  try {  ......  try {  return this.mappedFiles.get(index);  } catch (Exception e) {  ......  LOG\_ERROR.warn("findMappedFileByOffset failure. ", e);  }  }  } catch (Exception e) {  log.error("findMappedFileByOffset Exception", e);  }  }  }  LOG\_ERROR.warn("findMappedFileByOffset failure. ", e) and log.error("findMappedFileByOffset Exception", e) are different for recording exception messages.  LOG\_ERROR.error(..., e) should be used to record the exception message e. |
| **Taint flow path** | //source **<org.apache.rocketmq.store.ConsumeQueue: void recover()>** -- >  ......  <org.apache.rocketmq.client.impl.consumer.DefaultMQPushConsumerImpl: void start()> -- >  ......  <org.apache.rocketmq.store.ConsumeQueue: void recover()>  **<org.apache.rocketmq.store.MappedFileQueue: void truncateDirtyFiles(long)>**  //sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Apache Thrift |
| **Input** | Integration test:  We used its libraries to develop a calculator consisting of a server and a client component. (The Thrift file must be transferred to Java programs first.) We performed against the calculator (from its client) basic arithmetics (addition, subtraction, multiplication, and division). |
| **Description** | In org.apache.thrift.transport.TSaslClientTransport,  protected void handleSaslStartMessage() throws TTransportException, SaslException {  .......  LOGGER.debug("Sending mechanism name {} and initial response of length {}", mechanism,  initialResponse.length);  .......  }  Sensitive information about mechanism is leaked. The LOGGER.isDebugEnabled() conditional statement should be added. |
| **Taint flow path** | **<org.apache.thrift.transport.TIOStreamTransport: int read(byte[],int,int)>** --> // Source  <org.apache.thrift.transport.TTransport: int readAll(byte[],int,int)> -->  <org.apache.thrift.protocol.TBinaryProtocol: int readAll(byte[],int,int)> -->  ………  <org.apache.thrift.ProcessFunction: void process(int,org.apache.thrift.protocol.TProtocol,org.apache.thrift.protocol.TProtocol,java.lang.Object)> -->  <org.apache.thrift.transport.TIOStreamTransport: void flush()> -->  **<org.apache.thrift.transport.TSaslClientTransport: void handleSaslStartMessage()>** //Sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Apache Thrift |
| **Input** | Integration test:  We used its libraries to develop a calculator consisting of a server and a client component. (The Thrift file must be transferred to Java programs first.) We performed against the calculator (from its client) basic arithmetics (addition, subtraction, multiplication, and division). |
| **Description** | An information leakage from TIOStreamTransport to TSaslTransport  In org.apache.thrift.transport.TSaslTransport,  public void open() throws TTransportException {  .......  LOGGER.debug("{}: Start message handled", getRole());  .......  LOGGER.debug("{}: All done!", getRole());  .......  LOGGER.debug("{}: Main negotiation loop complete", getRole());  .......  LOGGER.debug("{}: SASL Client receiving last message", getRole());  .......  }  Sensitive information about Role is leaked. The LOGGER.isDebugEnabled() conditional statements should be added. |
| **Taint flow path** | **<org.apache.thrift.transport.TIOStreamTransport: int read(byte[],int,int)>** --> // Source  <org.apache.thrift.transport.TTransportInputStream: int read(byte[],int,int)> -->  <org.apache.thrift.transport.TTransport: int readAll(byte[],int,int)> -->  ………  <org.apache.thrift.transport.TIOStreamTransport: int read(byte[],int,int)> -->  <org.apache.thrift.protocol.TBinaryProtocol: void <init>(org.apache.thrift.transport.TTransport,long,long,boolean,boolean)> -->  <org.apache.thrift.protocol.TBinaryProtocol: org.apache.thrift.protocol.TMessage readMessageBegin()> -->  <org.apache.thrift.transport.TSocket: void <init>(java.net.Socket)> -->  <CalculatorClient: void main(java.lang.String[])> -->  **<org.apache.thrift.transport.TSocket: void open()>** //Sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Apache Thrift |
| **Input** | Integration test:  We used its libraries to develop a calculator consisting of a server and a client component. (The Thrift file must be transferred to Java programs first.) We performed against the calculator (from its client) basic arithmetics (addition, subtraction, multiplication, and division). |
| **Description** | Sensitive information about expected and actual reading lengths (len, got) is leaked from TIOStreamTransport to TTransport through a TtransportException.  In org.apache.thrift.transport.Ttransport:  public int readAll(byte[] buf, int off, int len)  throws TTransportException {  ......  if (ret <= 0) {  throw new TTransportException(  "Cannot read. Remote side has closed. Tried to read "  + len  + " bytes, but only got "  + got  + " bytes. (This is often indicative of an internal error on the server side. Please check your server logs.)");  }  ......  }  Sensitive information about expected and actual reading lengths (len, got) is leaked. |
| **Taint flow path** | **<org.xsocket.connection.IoConnector: void handleConnect()>** --> // Source  **<org.xsocket.connection.IoSocketDispatcherPool: org.xsocket.connection.IoSocketDispatcher nextDispatcher(int)>** //Sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Apache Thrift |
| **Input** | Integration test:  We used its libraries to develop a calculator consisting of a server and a client component. (The Thrift file must be transferred to Java programs first.) We performed against the calculator (from its client) basic arithmetics (addition, subtraction, multiplication, and division). |
| **Description** | Sensitive information about socket input stream or output stream is leaked from TIOStreamTransport.  In org.apache.thrift.transport.TIOStreamTransport,  public void close() {  if (inputStream\_ != null) {  try {  inputStream\_.close();  } catch (IOException iox) {  LOGGER.warn("Error closing input stream.", iox);  }  inputStream\_ = null;  }  if (outputStream\_ != null) {  try {  outputStream\_.close();  } catch (IOException iox) {  LOGGER.warn("Error closing output stream.", iox);  }  outputStream\_ = null;  }  }  Sensitive information about socket input stream or output stream is leaked. |
| **Taint flow path** | **<org.apache.thrift.transport.TIOStreamTransport: int read(byte[],int,int)>** --> // Source  <org.apache.thrift.transport.TTransport: int readAll(byte[],int,int)> -->  <org.apache.thrift.protocol.TBinaryProtocol: int readAll(byte[],int,int)> -->  ………  <org.apache.thrift.protocol.TBinaryProtocol: void writeI16(short)> -->  <org.apache.thrift.transport.TSocket: void close()> -->  <org.apache.thrift.protocol.TBinaryProtocol: void writeI16(short)> -->  **<org.apache.thrift.transport.TIOStreamTransport: void close()>** //Sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Apache Thrift |
| **Input** | Integration test:  We used its libraries to develop a calculator consisting of a server and a client component. (The Thrift file must be transferred to Java programs first.) We performed against the calculator (from its client) basic arithmetics (addition, subtraction, multiplication, and division). |
| **Description** | An information leakage about socket input stream or output stream from TIOStreamTransport to TSocket.  In org.apache.thrift.transport.TSocket:  public void close() {  ......  if (socket\_ != null) {  try {  socket\_.close();  } catch (IOException iox) {  LOGGER.warn("Could not close socket.", iox);  }  socket\_ = null;  }  }  Sensitive information about socket input stream or output stream is leaked. |
| **Taint flow path** | **<org.apache.thrift.transport.TIOStreamTransport: int read(byte[],int,int)>** --> // Source  <org.apache.thrift.transport.TTransport: int readAll(byte[],int,int)> -->  <org.apache.thrift.protocol.TBinaryProtocol: int readAll(byte[],int,int)> -->  ………  <org.apache.thrift.protocol.TBinaryProtocol: void writeMessageBegin(org.apache.thrift.protocol.TMessage)> -->  <CalculatorClient: void main(java.lang.String[])> -->  <CalculatorService$add\_result: boolean isSetSuccess()> -->  <org.apache.thrift.protocol.TBinaryProtocol: void writeI16(short)> -->  **<org.apache.thrift.transport.TSocket: void close()>** //Sink |
| **Status** | Confirmed |

|  |  |
| --- | --- |
| **Subject** | Voldemort |
| **Input** | System test in the project package:  During voldemort system testing, we started one server and two clients, then we executed "voldemort.partition.TestDistribution" |
| **Description** | No verification for illegal values of the parameter "protocolVersion" in VoldemortNativeClientRequestFormat  voldemort.client.protocol.vold.VoldemortNativeClientRequestFormat:  public VoldemortNativeClientRequestFormat(int protocolVersion) {  this.mapper = new ErrorCodeMapper();  this.protocolVersion = protocolVersion;  }  There is no protocolVersion verification. The value of protocolVersion must be 0,1,2 or 3 so that we need to verifiy its value. |
| **Taint flow path** | **<voldemort.store.configuration.FileBackedCachingStorageEngine: void loadData()**> -->  // Source  <voldemort.server.RequestRoutingType: void <clinit>()> -->  <voldemort.server.RequestRoutingType: voldemort.server.RequestRoutingType getRequestRoutingType(boolean,boolean)> -->  <voldemort.client.protocol.RequestFormatType: java.lang.String getCode()> -->  <voldemort.client.protocol.RequestFormatType: voldemort.client.protocol.RequestFormatType fromCode(java.lang.String)> -->  <voldemort.cluster.Zone: int getId()> -->  <voldemort.cluster.Node: java.util.List getPartitionIds()> -->  …….  <voldemort.cluster.Node: int getRestPort()> -->  <voldemort.cluster.Node: int getZoneId()> -->  <voldemort.cluster.Node: int getSocketPort()> -->  <voldemort.common.service.AbstractService: boolean isStarted()> -->  <voldemort.store.metadata.MetadataStore$VoldemortState: voldemort.store.metadata.MetadataStore$VoldemortState valueOf(java.lang.String)> -->  <voldemort.store.metadata.MetadataStore: voldemort.store.metadata.MetadataStore$VoldemortState getServerStateUnlocked()> -->  <voldemort.client.RoutingTier: java.lang.String toDisplay()> -->  <voldemort.store.system.SystemStoreConstants: boolean isSystemStore(java.lang.String)> -->  **<voldemort.client.protocol.vold.VoldemortNativeClientRequestFormat: void <init>(int)>**  //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | Voldemort |
| **Input** | System test in the project package:  During voldemort system testing, we started one server and two clients, then we executed "voldemort.partition.TestDistribution" |
| **Description** | An information leakage from FileBackedCachingStorageEngine to ClientRequestExecutor without control through an EOFException  voldemort.store.socket.clientrequest.ClientRequestExecutor:  protected void read(SelectionKey selectionKey) {  ......  if((count = socketChannel.read(inputStream.getBuffer())) == -1)  throw new EOFException("EOF for " + socketChannel.socket());  ......  } |
| **Taint flow path** | **<voldemort.store.configuration.FileBackedCachingStorageEngine: void loadData()>** --> // Source  <voldemort.server.RequestRoutingType: void <clinit>()> -->  <voldemort.server.RequestRoutingType: voldemort.server.RequestRoutingType getRequestRoutingType(boolean,boolean)> -->  <voldemort.client.protocol.RequestFormatType: java.lang.String getCode()> -->  <voldemort.client.protocol.RequestFormatType: voldemort.client.protocol.RequestFormatType fromCode(java.lang.String)> -->  <voldemort.cluster.Zone: int getId()> -->  <voldemort.cluster.Node: java.util.List getPartitionIds()> -->  <voldemort.versioning.VectorClockUtils: voldemort.versioning.Occurred compare(voldemort.versioning.VectorClock,voldemort.versioning.VectorClock)> -->  <voldemort.versioning.VectorClock: voldemort.versioning.Occurred compare(voldemort.versioning.Version)> -->  <voldemort.cluster.Cluster: java.util.Collection getNodesShuffled()> -->  <voldemort.routing.RouteToAllStrategy: void <init>(java.util.Collection)> -->  <voldemort.cluster.Node: java.lang.String getHost()> -->  <voldemort.cluster.failuredetector.FailureDetectorConfig: voldemort.cluster.Cluster getCluster()> -->  <voldemort.cluster.failuredetector.AbstractFailureDetector: voldemort.cluster.failuredetector.FailureDetectorConfig getConfig()> -->  <voldemort.cluster.failuredetector.FailureDetectorConfig: voldemort.utils.Time getTime()> -->  <voldemort.cluster.failuredetector.AsyncRecoveryFailureDetector: void run()> -->  <voldemort.utils.SystemTime: void sleep(long)> -->  <voldemort.cluster.Node: int getAdminPort()> -->  ……  <init>(voldemort.server.protocol.RequestHandlerFactory,int,int,boolean,int,java.lang.String,boolean,int,long)> -->  <voldemort.cluster.failuredetector.AbstractFailureDetector: void addFailureDetectorListener(voldemort.cluster.failuredetector.FailureDetectorListener)> -->  <voldemort.store.bdb.BdbRuntimeConfig: boolean isCheckpointerOffForBatchWrites()> -->  <voldemort.common.nio.ByteBufferBackedOutputStream: java.nio.ByteBuffer getBuffer()> -->  <voldemort.store.bdb.BdbRuntimeConfig: boolean getMinimizeScanImpact()> -->  <voldemort.common.service.SchedulerService: void <init>(int,voldemort.utils.Time,boolean)> -->  <voldemort.store.socket.clientrequest.ClientRequestExecutor: void write(java.nio.channels.SelectionKey)> -->  **<voldemort.store.socket.clientrequest.ClientRequestExecutor: void read(java.nio.channels.SelectionKey)>** //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | Voldemort |
| **Input** | System test in the project package:  During voldemort system testing, we started one server and two clients, then we executed "voldemort.partition.TestDistribution" |
| **Description** | Sensitive information about socketChannel socket is leaked from FileBackedCachingStorageEngine to SelectorManagerWorker  voldemort.common.nio.SelectorManagerWorker,  public void run() {  try {  ......  } catch(IOException e) {  logger.info("Connection reset from " + socketChannel.socket() + " with message - "  + e.getMessage());  close();  } |
| **Taint flow path** | **<voldemort.store.configuration.FileBackedCachingStorageEngine: void loadData()>** -->  // Source  <voldemort.server.RequestRoutingType: void <clinit>()> -->  <voldemort.server.RequestRoutingType: voldemort.server.RequestRoutingType getRequestRoutingType(boolean,boolean)> -->  <voldemort.client.protocol.RequestFormatType: java.lang.String getCode()> -->  <voldemort.client.protocol.RequestFormatType: voldemort.client.protocol.RequestFormatType fromCode(java.lang.String)> -->  <voldemort.cluster.Zone: int getId()> -->  <voldemort.cluster.Node: java.util.List getPartitionIds()> -->  <voldemort.versioning.VectorClockUtils: voldemort.versioning.Occurred compare(voldemort.versioning.VectorClock,voldemort.versioning.VectorClock)> -->  <voldemort.versioning.VectorClock: voldemort.versioning.Occurred compare(voldemort.versioning.Version)> -->  <voldemort.cluster.Cluster: java.util.Collection getNodesShuffled()> -->  <voldemort.routing.RouteToAllStrategy: void <init>(java.util.Collection)> -->  <voldemort.cluster.Node: java.lang.String getHost()> -->  <voldemort.cluster.failuredetector.FailureDetectorConfig: voldemort.cluster.Cluster getCluster()> -->  <voldemort.cluster.failuredetector.AbstractFailureDetector: voldemort.cluster.failuredetector.FailureDetectorConfig getConfig()> -->  <voldemort.cluster.failuredetector.FailureDetectorConfig: voldemort.utils.Time getTime()> -->  <voldemort.cluster.failuredetector.AsyncRecoveryFailureDetector: void run()> -->  <voldemort.utils.SystemTime: void sleep(long)> -->  <voldemort.cluster.Node: int getAdminPort()> -->  ……  <init>(voldemort.server.protocol.RequestHandlerFactory,int,int,boolean,int,java.lang.String,boolean,int,long)> -->  <voldemort.cluster.failuredetector.AbstractFailureDetector: void addFailureDetectorListener(voldemort.cluster.failuredetector.FailureDetectorListener)> -->  <voldemort.store.bdb.BdbRuntimeConfig: boolean isCheckpointerOffForBatchWrites()> -->  <voldemort.common.nio.ByteBufferBackedOutputStream: java.nio.ByteBuffer getBuffer()> -->  <voldemort.store.bdb.BdbRuntimeConfig: boolean getMinimizeScanImpact()> -->  <voldemort.common.service.SchedulerService: void <init>(int,voldemort.utils.Time,boolean)> -->  <voldemort.store.socket.clientrequest.ClientRequestExecutor: void write(java.nio.channels.SelectionKey)> -->  <voldemort.store.socket.clientrequest.ClientRequestExecutor: void read(java.nio.channels.SelectionKey)> --> //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | Voldemort |
| **Input** | System test in the project package:  During voldemort system testing, we started one server and two clients, then we executed "voldemort.partition.TestDistribution" |
| **Description** | Sensitive information about socketChannel socket is leaked from from FileBackedCachingStorageEngine to ClientRequestExecutorFactory  voldemort.store.socket.clientrequest.ClientRequestExecutorFactory:  public ClientRequestExecutor create(SocketDestination dest) {  ......  if(socketChannel.socket().getReceiveBufferSize() != this.socketBufferSize)  logger.debug("Requested socket receive buffer size was " + this.socketBufferSize  + " bytes but actual size is "  + socketChannel.socket().getReceiveBufferSize() + " bytes.");  ......  } |
| **Taint flow path** | **<voldemort.store.configuration.FileBackedCachingStorageEngine: void loadData()>** --> // Source  <voldemort.server.RequestRoutingType: void <clinit>()> -->  <voldemort.server.RequestRoutingType: voldemort.server.RequestRoutingType getRequestRoutingType(boolean,boolean)> -->  <voldemort.client.protocol.RequestFormatType: java.lang.String getCode()> -->  <voldemort.client.protocol.RequestFormatType: voldemort.client.protocol.RequestFormatType fromCode(java.lang.String)> -->  <voldemort.cluster.Zone: int getId()> -->  <voldemort.cluster.Node: java.util.List getPartitionIds()> -->  <voldemort.versioning.VectorClockUtils: voldemort.versioning.Occurred compare(voldemort.versioning.VectorClock,voldemort.versioning.VectorClock)> -->  <voldemort.versioning.VectorClock: voldemort.versioning.Occurred compare(voldemort.versioning.Version)> -->  <voldemort.cluster.Cluster: java.util.Collection getNodesShuffled()> -->  <voldemort.routing.RouteToAllStrategy: void <init>(java.util.Collection)> -->  <voldemort.cluster.Node: java.lang.String getHost()> -->  <voldemort.cluster.failuredetector.FailureDetectorConfig: voldemort.cluster.Cluster getCluster()> -->  <voldemort.cluster.failuredetector.AbstractFailureDetector: voldemort.cluster.failuredetector.FailureDetectorConfig getConfig()> -->  <voldemort.cluster.failuredetector.FailureDetectorConfig: voldemort.utils.Time getTime()> -->  <voldemort.cluster.failuredetector.AsyncRecoveryFailureDetector: void run()> -->  <voldemort.utils.SystemTime: void sleep(long)> -->  <voldemort.cluster.Node: int getAdminPort()> -->  ……  <voldemort.common.nio.ByteBufferContainer: void updateSizeStats(int,int)> -->  <voldemort.common.nio.ByteBufferContainer: void assignBuffer(java.nio.ByteBuffer)> -->  <voldemort.common.nio.ByteBufferContainer: void <init>(int,int,org.apache.commons.lang.mutable.MutableLong)> -->  <voldemort.common.nio.ByteBufferBackedInputStream: void <init>(voldemort.common.nio.ByteBufferContainer)> -->  <voldemort.common.nio.ByteBufferBackedOutputStream: void <init>(voldemort.common.nio.ByteBufferContainer)> -->  <voldemort.store.socket.clientrequest.ClientRequestExecutor: void initializeStreams(int,voldemort.common.nio.CommBufferSizeStats)> -->  <voldemort.store.socket.clientrequest.ClientRequestExecutor: void <init>(java.nio.channels.Selector,java.nio.channels.SocketChannel,int,long,voldemort.store.socket.SocketDestination)> -->  **<voldemort.store.socket.clientrequest.ClientRequestExecutorFactory$1: void <init>(voldemort.store.socket.clientrequest.ClientRequestExecutorFactory,voldemort.store.socket.SocketDestination,voldemort.utils.pool.KeyedResourcePool)>** //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | xSocket |
| **Input** | Integration test:  Two nodes were started and then sends messages to another node. |
| **Description** | An information leakage from IoConnector to IoSocketDispatcher through an IOException without LOG severe control:  org.xsocket.connection:  public IoSocketDispatcher(AbstractMemoryManager memoryManager, String name) {  ......  try {  selector = Selector.open();  } catch (IOException ioe) {  String text = "exception occured while opening selector. Reason: " + ioe.toString();  LOG.severe(text);  throw new RuntimeException(text, ioe);  }  ......  }  The statement "LOG.severe(text)" doesn't have LOG severe control. |
| **Taint flow path** | **<org.xsocket.connection.IoConnector: void handleConnect()>** --> // Source  <org.xsocket.connection.IoConnector$RegisterTask: org.xsocket.connection.IIoConnectorCallback access$100(org.xsocket.connection.IoConnector$RegisterTask)> -->  <org.xsocket.connection.IoConnector: void handleConnect()> -->  <org.xsocket.connection.ConnectionManager: org.xsocket.connection.ConnectionManager$TimeoutMgmHandle  register(org.xsocket.connection.NonBlockingConnection)> -->  …….  <org.xsocket.connection.NonBlockingConnection: org.xsocket.connection.IoChainableHandler createClientIoHandler(java.nio.channels.SocketChannel,javax.net.ssl.SSLContext,boolean)> -->  <org.xsocket.connection.IoSocketDispatcherPool: org.xsocket.connection.IoSocketDispatcher nextDispatcher(int)> //Sink |
| **Status** | Pending developer confirmation |

|  |  |
| --- | --- |
| **Subject** | Apache Zookeeper |
| **Input** | Load test in the project package:  During the system testing, we started a Zookeeper server, then we executed "org.apache.zookeeper.test.system.InstanceContainer" to open a Zookeeper container. Lastly, we performed "org.apache.zookeeper.test.system.GenerateLoad". |
| **Description** | In methods of org.apache.zookeeper.test.system.GenerateLoad,  public static void main(String[] args) throws InterruptedException,  KeeperException, NoAvailableContainers, DuplicateNameException,  NoAssignmentException {  ......  if (!statusWatcher.waitConnected(5000)) {  System.err.println("Could not connect to " + args[0]);  return;  }  ......  String mode = getMode(parts[i]);  if (mode.equals("leader")) {  zkHostPort = new StringBuilder(parts[i]);  System.out.println("Connecting exclusively to " + zkHostPort.toString());  break outer;  }  try {  String cmdNumber[] = line.split(" ");  ......  } catch (NumberFormatException e) {  System.out.println("Not a valid number: "  + e.getMessage());  }  ......  }  }  Sensitive data about args[0], zkHostPort, and cmdNumber are directly printed and may leak.  For security, log should be used to record these data, as well as log in other classes such as org.apache.zookeeper.server.ZooKeeperServer:  LOG = LoggerFactory.getLogger(GenerateLoad.class);  ......  LOG.error("Could not connect to " + args[0]);  ......  LOG.info("Connecting exclusively to " + zkHostPort.toString());  ......  LOG.error("Not a valid number: " + e.getMessage()); |
| **Taint flow path** | //source **<org.apache.zookeeper.test.system.GenerateLoad: void main(java.lang.String[])>**  ......  **<org.apache.zookeeper.test.system.GenerateLoad: void main(java.lang.String[])>** //sink |
| **Status** | Fixed |