Secure Encapsulation of Insecure Middleware

ZHAO Jian-min[1] **ZHANG Wei**[2] **LI Tian-gei**[3]

[1] Daqing Petroleum Institute, Daqing, Heilongjiang, 163318, China
[2]Oil Recovery Plant No.9 of Daqing Oilfield Corp.Ltd., Daqing, 163853, China
[3] Natural Gas Sub-company of Daqing Oilfield Corp. Ltd., Daqing, 163459, China

Abstract

Unceasingly developed and deeply exploited along with the software system, the variety of middlewares becomes more and more, they mutually affect each other in the complex way. These middlewares, because of the complex of the origin, are not extremely credible. So, before using these middlewares, we must understand the security features of them, for instance, the secret data cannot be leaked in the network. But, it is very difficult to confirm whether these middlewares have good security feature. The paper designs the encapsulations, which let these middlewares run under secure environment, and it provides the control of good granularity among the middlewares, the middleware and other system resources. The main part of this paper is to research the expression methods of the encapsulations, and we state and verify the security through these methods. This paper uses box- calculus to describe several kinds of encapsulations, and discuss the security that each encapsulation could guarantee.

Keywords: Middleware, Encapsulation, Security, Calculus

1. Introduction

The software system develops unceasingly, more and more sole applications are replaced by software middlewares which come from different origins. Now, widely, the distributed application systems are all constructed by some small middlewares, they mutually affect each other in the complex way, execute various information processing task. Furthermore, although middleware base doesn't change, and system administrator often controls middleware base, it is easy to download source code in the network; some technology even allows dynamic use new middleware during the course of program running.

Under such variable environment of operation, the traditional security mechanism and the strategy appear extremely draggle. Though the password and the access control mechanism suit to protect the integrity of system, it can't solve the problem that user download current running code. Some methods (for example Java sand box) promise the security through isolation. But, these methods are also unsatisfying, because the middleware can mutually affect each other freely, or each other do not mutually affect radically. So we need a kind of extremely good protection mechanism, which can control the mutual connection between the middlewares.

Although it is not easy to analyze and modify large-scale and the third party software package, it can prevent correspondences between the software package and other parts of the system, distill the code of different software middleware boundary. So, it can monitor the transfer operation and commutative data among these middlewares. This paper entitles the security encapsulation which can encapsulate the code frame of the incredible middlewares.

Obviously, writing to encapsulation cannot leave End-User simply, user only to choose the most appropriate encapsulation, make its parameter and install it. All these processes are dynamic: comparing with the new application procedure, the encapsulation should be easier to join to the movement system. The user needs the encapsulation to be able to guarantee a secure clear description.

This paper main researches security environment of encapsulation, discusses with emphasis that how to express the encapsulation, and can define strictly and prove it. Obviously, if there is not such strict definition and verifying, it is difficult for the designer to study thoroughly. Although the encapsulation is very important, it is possibly small software, therefore it is very easy to prove its attribute.

2. Safety encapsulation

This paper designs four encapsulations. The first encapsulation has encapsulated an independent middleware, limits it's correlation with outside and only follows the specific protocol to be able to correspond with outside processes. The second encapsulation is very similar with the first, only has recorded log of all correspondences. The third encapsulation has encapsulated two middlewares, it allows each middleware to interact with outside through the definition way, and the information of the first middleware may transmit to the second middleware. The fourth encapsulation has encapsulated three middlewares, and has controlled the interactive between it and environment, has limited these correlations which merely have realized through channel in and out, has achieved the secure goal.

The design of an encapsulation must connect with the transmission protocols which use in some middlewares and environment or among middlewares. Regarding the first two encapsulations, the paper has fixed two channels, *in* and *out*. They are all independent in receiving and transmitting information. Besides, here supposed middleware can execute in several independent boxes. The received values *v*, we make its copy, and make a pair < y y > and output it. This can write as:

$$!in^{\uparrow} y.\overline{out}^{\uparrow} < y \;\; y >$$

A wrong middleware can also import data to an illegal output channel. For example:

$$!in^{\uparrow} y.(\overline{net}^{\uparrow} y \mid \overline{out}^{\uparrow} < y \;\; y >)$$

Or monitor the transmissions in other parts of this system, for example:

$$!c^{*} y.(\overline{net}^{\uparrow} c \mid \overline{c}^{*} y)$$

When we describe a middleware whether follows adding label to transforms the semantics, for unitary encapsulation *P* may operate normally, when only $\mathrm{A} \mid\!- \;\; \mathrm{P} \xrightarrow{l_1 \ldots l_k} Q$, then l_j is $in^{\uparrow} v$, $\overline{out}^{\uparrow} v$ or τ .

3. Filter encapsulation

Filter is referred as the encapsulation which can purely limit the capacity of traffic. Considering a static filter encapsulation which only interactive between *in* and *out* channels.

Install a massager; it can cross the boundary to transfer legitimate information, executing middlewares in a new naming box throughout the process W_1. Notice that the relation of W1 and deep encapsulation isn't binding, it is equal.

Supposing, anywhere apply *W₁* to a process P, cannot freely bring new binding a in P. Do not consider the performance of P, *W₁* [P] must follow one kind of protocol that can describe clearly through marking transmission primitive.

Proposition 1 For any process P, $a \notin fn(P)$, if $A \mid\!- \; W_1[P] \xrightarrow{l_1 \ldots l_k} Q$ then l_j is $in^{\uparrow} v, \overline{out}^{\uparrow} v$ or τ form.

We can prove it through obtaining direct description of the condition. But this condition can be obtained by adding marking transform *W₁* [P]. This characteristic of unitary encapsulation is especially abstract.

4. The log encapsulation

Filter transmits the transcription periphery through *log* channel, and keeps all correspondences log in the process.

$$L[\_] \overset{def}{=} (va)(\; a[\_] \;|\; !in^{\uparrow} y.(\overline{\log}^{\uparrow} y \mid \overline{in}^{a} y) \;|\; out^{a} y.(\overline{\log}^{\uparrow} y \mid \overline{out}^{\uparrow} y)$$

An encapsulation middleware $L[P]$ also can alternate through restricted ways once more.

Proposition 2 For all middleware P and $a \notin fn(P)$, if $A \mid\!- \; L[P] \xrightarrow{l_1 \ldots l_n} Q$, then l_j is $in^{\uparrow} v$, $\overline{out}^{\uparrow} v$, $\overline{\log}^{\uparrow} v$ or τ.

5. Pipeline encapsulation

Pipeline encapsulation can control restrained information flow between two middlewares. Here gives duality encapsulation *W₂*, which contains two pieces of processes. There are two middlewares Q_i which are encapsulated in $W_2[Q_1, Q_2]$, and can intact with environment with channel *inᵢ* and *outᵢ*, moreover, Q_1 can transmit messages to Q_2 through *mid* channel. Here the execution of pipeline is disorder.

$$W_2[\_{}_1, \_{}_2] \overset{def}{=} (va_1, a_2)(\; a_1[\_{}_1] \mid a_2[\_{}_2] \;|\; !in_1^{\uparrow} y.\overline{in_1}^{a_1} y \;|\; !in_2^{\uparrow} y.\overline{in_2}^{a_2} y \;|\; out_1^{a_1} y.\overline{out_1}^{\uparrow} y) \;|\; out_2^{a_2} y.\overline{out_2}^{\uparrow} y) \;|\; mid_1^{a_1} y.\overline{mid_1}^{a_2} y)$$

Likewise, when *W₂* was not bounded, we always apply *W₂* to processes P_1, P_2, and suppose $\{a_1, a_2\} \cap fn(P_1, P_2) = \phi$. If only it satisfies the suitable free name for any process, this is $\{a_1, a_2\} \cap fn(P_1, P_2) = \phi$, then we think that the duality encapsulate is true, if $A \mid\!- \; L[P_1, P_2] \xrightarrow{l_1 \ldots l_n} Q$, then l_j is $in_i^{\uparrow} v$, $\overline{out_i}^{\uparrow} v$ or τ form.

Proposition 3 *W₂* is true.

For instance, suppose, $P_2 = \overline{mid}^{\uparrow} v$, the second encapsulated process transmits a data to the first process.

$$W_2[P_1, \overline{mid}^{\uparrow} v] = (va_1, a_2)(\; a_1[P_1] \mid a_2[\overline{mid}^{\uparrow} v] \mid R)$$
$$\rightarrow (va_1, a_2)(\; a_1[P_1] \mid a_2[0] \mid \overline{mid}^{\overline{a_2}} v \mid R)$$

There R is an integration of parallel transmitting. The outputting of $\overline{mid}^{\overline{a_2}} v$ can't carry on deeper interaction under the decisive conditions, so when a_2 is limited, it can't hand over more deeper interaction with each other surroundings the environment, also, when $a_1 \neq a_2$, it can't deliver $!mid^{a_1} y.\overline{mid}^{a_2} y$.

These encapsulations are supposed with a simple and fixed protocol. It directly produces a series arbitrarily channel to replace *in*, *out* and *mid*s, and it also directly allows n-encapsulation to encapsulate a lot of middlewares, which makes information transmitted according to the given order among the middlewares.

6. Ternary encapsulation

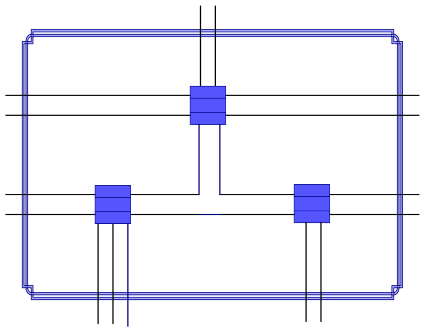

Figure 6.1 ternary encapsulation

Figure 6.1 showed us encapsulation *W* which encapsulates three middlewares *M₁*, *M₂*, and *M₃*, and it controls the interactions with environment, and limits these interactions which carry out through the channel *in* and *out* to get the security purpose. *M₁*, *M₂*, and *M₃* are all connected with *net*; *M₃* is also connected with *open window*.

We define a ternary encapsulation *W* to encapsulate middlewares, which are named *M₁*, *M₂*, and *M₃*, so produce three boxes which named a₁, a₂, and a₃, they have two transmission channels--one receives information through the channel *in* from environment, and send information to the encapsulation process; another receives information from encapsulation process through the channel *out*, and sends information to environment. The *M₁* still has a log channel.

$|!\ in_1^{y} \cdot (\overline{in}^{a_1} y \mid \overline{\log}^{\uparrow} y)$

The sensor receives information from environment through channel *in₁*, and sends information to the encapsulation process of *a₁*, and records the communication through the *log* channel.

$|!\ out_1^{a_1} y \cdot (\overline{out_1}^{\uparrow} y \mid \overline{\log}^{\uparrow} y)$

The sensor receives information from the encapsulation process of *a₁* and sends information to environment, and records the communication through the *log* channel.

$|!\ in_2^{\uparrow} \cdot \overline{in}^{a_2} y$

The sensor receives information from environment through *in₂* and sends information to the encapsulation process of *a₂*.

$|!\ out_2^{a_2} y \cdot \overline{out_2}^{\uparrow} y$

The sensor receives information from the encapsulation process of *a₂* through the channel *out₂* and sends information into environment.

$|!\ in_3^{\uparrow} \cdot \overline{in_3}^{a_3} y$

The sensor receives information from environment through *in₃* and sends information to the encapsulation process of *a₃*.

$|!\ out_3^{a_3} y \cdot \overline{out_3}^{\uparrow} y$

The sensor receives information from the encapsulation process of *a₃* through the channel *out₃* and sends information to environment.

|!

$net\_in_1^{\uparrow} y \cdot (\overline{net\_in_1}^{a_1} y \mid \overline{\log}^{\uparrow} y)$

The sensor receives information from *net* through channel *in₁*, and sends information to the encapsulation process of a₁, then records the communication through the *log* channel.

$|!\ net\_out_1^{a_1} y \cdot$

In3 out3

Net M3

Op
wind

Mid13 Mid23

Net M1 M2 Ne

Mid12

In1 out1 log In2 out2

The sensor receives information from the encapsulation process of *a₁* through channel *in₁*, and sends information to *net* if it did not contain personal information such as the E-mail, and then records the communication through *log* channel, or records the communication as empty process.

|! $net\_in_2^{\uparrow} y \cdot\ if$ y doesn't come from some domain or IP $then\ \ \overline{net\_in_2}^{a_2} y$ $else\ 0$.

The sensor receives information from *net* through channel *in₂*, and sends information to the encapsulation process of a_2 when the information *y* doesn't come from some domain or IP, or records the communication as empty process.

|! $net\_out_3^{a_3} y \cdot \overline{net\_out_3}^{\uparrow} y$

The sensor receives information from the encapsulation process of a_3 through channel *out₃*, and sends information to *net*.

|! $net\_in_3^{\uparrow} y \cdot\ if$ $y \in \{update, install\}\ \ then$ $\overline{net\_in_3}^{a3} y\ \ else\ 0$

The sensor receives information from *net* through channel *in₃*, and sends information to the encapsulation process of a_3 when *y updates* or *installs* information, or records the communication as empty process.

|! $net\_out_3^{a_3} y \cdot \overline{net\_out_3}^{\uparrow} y$

The sensor receives information from the encapsulation process of a_3 through channel *out₃*, and sends the information to *net*.

|! $mid_{13}^{a_1} y \cdot\ (if\ \ \ y \in \{0,1\}\ \ then$ $\overline{mid_{13}}^{a_3} y\ \ else\ \ 0 \mid \overline{\log}^{\uparrow} y)$

M_1 sends information *y* of encapsulation process of a_1 to M_3 through channel mid_{13}, when *y* is 0 or 1, sends information of encapsulation process of M_3 to M_1, otherwise, records the communication as empty process.

|!

$mid_{12}^{a_1} y \cdot (\overline{mid\ \ _1}^{a_2} y \mid \overline{\log}^{\uparrow} y)$

The sensor exchanges information between M_1 and M_3 through channel mid_{13}, and records the communication through *log* channel.

|!

$mid_{12}^{a_1} y \cdot (\overline{mid\ \ _1}^{a_2} y \mid \overline{\log}^{\uparrow} y)$

The sensor exchanges information between M_1 and M_2 through channel mid_{12}, and records the communication through *log* channel.

|! $mid_{23}^{a_2} y \cdot \overline{mid_{23}}^{a_3} y$

The sensor exchanges information between M_3 and M_2 through channel mid_{23}.

|! $openwindow^{a_3}(s\ \ \ x) \cdot \overline{openwindow}^{\uparrow} < s\ \ \ x >$

$\mid x^{\uparrow}(getc\ \ \ putc\ \ \ close) \cdot \overline{x}^{a_3} < getc\ \ \ putc\ \ \ close >$

|! $getc^{a_3} y \cdot (\overline{getc}^{\uparrow} < c\ \ \ y > \mid y^{\uparrow} \overline{y}^{a_3} c)$

|! $putc^{a_3}(c\ \ \ y) \cdot (\overline{putc}^{\uparrow} < c\ \ \ y > \mid y^{\uparrow} \overline{y}^{a_3} \cdot\ )$

|! $close^{a_3} y \cdot (\overline{close}^{\uparrow} y \mid y^{\uparrow} \cdot \overline{y}^{a_3})$

The sensor receives input information *x* that comes from any son box through channel *s*, and binds the name of the son box to a_3. It can also send output information *x* of a_3 to *open window*.

The information *x* of *open window* can be read , written and closed into M_3, and can also be read , written and closed into a_3.

The sensor reads the information *y* coming from a_3 and reads it into the *open window*, then deliver the received y to a3.

The sensor sends the information *y* coming from *c* into a_3. Then deliver the information y to a3.

The sensor closes the information *y* to a_3, and then sends to a3.

This model shows an encapsulation which encapsulates three middlewares, it controls the interactions among any middlewares, the interactions between the middlewares and environment, and the interactions between the middlewares and *net*. It limits the interactions between the middlewares and *net* just through the channel *in* and *out*. By instancing the connection of M_1 to *log*, M_3 to *open window*, it obtained the security purpose.

7 Conclusions and further work

This paper puts forward a set of theories that used for the secure encapsulation of middlewares. It designs and proves four encapsulations: the filter encapsulation, the log encapsulation, the pipeline encapsulation and the ternary middlewares encapsulation. It is easy to make n-encapsulation according to the ternary middlewares encapsulation. These encapsulations can keep information exchanging secure among an insecure middleware with other middlewares, the network, the operate system, the run-time and the log etc. And this model can implement dynamic and flexible security strategy.

This paper develops many directions that worth our deep research, and a lot of deductions and conclusion are waited for further mining and exploring, also requests further effort to comprehend about binary system encapsulation. At present, the four encapsulations haven't yet been achieved practical applied and deployment, they are still in the theoretical stage.

8. References

[1] Nayeem Islam, Rangachari Anand, Trent Jaeger, and Josyula R. Rao (1997). “A flexible security system for using Internet content”. IEEE Software, 14(5):52-59.

[2] Andrew C. Myers (1999). “Jflow Practical static information flow control”. In Proceedings of the 26th ACM Symposium on Principles of Programming Languages.

[3] Peter Sewell (1997). “Global local subtyping for a distributed -calculus”. Technical Report 435, University of Cambridge.

[4] Peter Sewell (1998). “Global local sub typing and capability inference for a distributed -calculus”. In Proceedings of ICALP’98, LNCS 1443, pp. 695-706.

[5] Peter Sewell (1999). “A brief introduction to applied”, Lecture notes for the Mathfit Instructional Meeting on Recent Advances in Semantics and Types for Concurrency: Theory and Practice.

[6] [6] D. Volpano, C. Irvine, and G. Smith (1996). “A sound type system for secure flow analysis”. Journal of Computer Security, pp.167-187.

[7] C.Boutilier, R.Reiter, and B.Price (2001). “Symbolic dynamic programming for first-order MDPs”. In B.Nebel, editor, Proceedings of the Seventeenth International Conference on Artificial Intelligence (IJCAI-01), pp. 690-700.

[8] C.Boutiliter, R.Reiter, M.Soutchanski, and S.Thrun (2000). “Decision-theoretic, high-level agent programming in the situation calculus”. In proceedings of the Seventeenth International Conference on Artificial Intelligence (AAAI-00).