Chapter 4
Process Calculus for Cost Analysis of Process Creation

Shin-ya Nishizaki, Mizuki Fujii and Ritsuya Ikeda

Abstract Many researchers have proposed formal frameworks for analyzing cryptographic and communication protocols and have studied the theoretical properties of the protocols, such as authenticity and secrecy. The resistance of denial-of-service attacks is one of the most important issues in network security. Several researchers have applied process calculi to security issues. One of the most remarkable of these studies is Abadi and Gordon's, based on Milner's pi-calculus. For the denial-of-service attack, Medow proposed a formal framework based on the Alice-and-Bob notation, and Tomioka et al. proposed a process calculus based on Milner's pi-calculus, the Spice-calculus. Using the Spice-calculus, we can evaluate the computational cost of executing processes. In our previous works, the Spice-calculus could analyze computational costs such as arithmetic operations, hash computation, and message transmission. However, the cost of process creation was disregarded. In this paper, we improve the Spice-calculus through adding cost evaluation of process creations. We extend the syntax of the cost in the Spice-calculus and operational semantics of the Spice-calculus. We then present an example of the improved Spice-calculus.

Keywords Denial-of-service · Attack · Process calculus · Pi-calculus

S. Nishizaki (✉) · M. Fujii · R. Ikeda
Department of Computer Science, Tokyo Institute of Technology, 2-12-1-W8-69, Ookayama, Meguro-ku, Tokyo 152-8552, Japan
e-mail: nisizaki@cs.titech.ac.jp

M. Fujii
e-mail: mizuki.fujii@lambda.cs.titech.ac.jp

R. Ikeda
e-mail: ritsuya.ikeda@lambda.cs.titech.ac.jp

4.1 Introduction

Vulnerability of communication protocols can cause various kinds of attacks, which cause substantial damage to systems connected to the Internet. A *denial-of-service attack* is one of the constant concerns of computer security. It can make the usual services of a computer or network inaccessible to the regular users. An archetypal example of DoS attack is a *SYN flooding attack* [1] on the Transmission Control Protocol (TCP) (Fig. 4.1).

Before sending data from a source host S to a destination host D, the hosts have to establish a connection between S and D, called *three-way handshake*. The host S begins by sending a SYN packet including an initial sequence number x. The host D then replies to S with a message in which the SYN and ACK flags are set, indicating that D acknowledges the SYN packet from the S. The message includes D's sequence number y and incremented S's sequence number ($x + 1$) as an ACK number. The host S sends a message with an ACK bit, a SEQ number ($x + 1$), and an ACK number ($y + 1$).

Consider a situation that lasts for a short period during which an attacker, A, sends an enormous number of connection requests with spoofed source IP-addresses to a victim host, D. The number of actual implementations of half-opened connections per port is limited since the memory allocation of data structures for such a connection is limited (Fig. 4.2).

Recently, several researchers have studied DoS attacks from various view-points, such as protection from DoS attacks using resource monitoring [2] and the development of DoS resistance analysis [3, 4]. She extended the Alice-and-Bob notation by attaching an atomic procedure annotation to each communication.

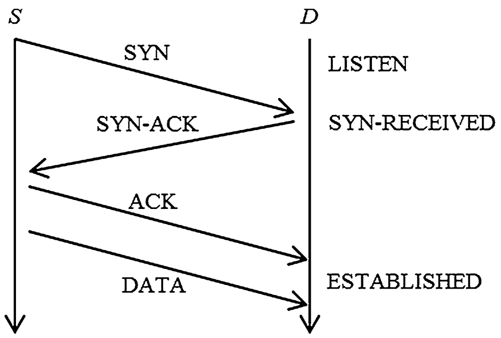
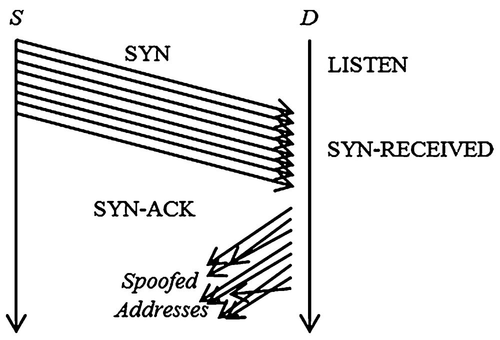


Fig. 4.1 Three-way handshake of TCP

Fig. 4.2 SYN-flood attack



4.2 Process Calculus for Analyzing Denial-of-Service Attacks

4.2.1 The Spice Calculus

Tomioka et al. [5] proposed a formal system for DoS resistance analysis, the Spice calculus, which is based on Milner's process calculus [6] and its extension for secure computation [7]. A characteristic feature of the calculus is a type system which enables us to track the computational cost of each computer node. The type of the Spice calculus represents the configuration of a distributed system.

For example, a typing of the Spice-calculus

$$\mathcal{A}|(\mathcal{B}|\mathcal{C}) \triangleright P_1|(Q|P_2)$$

means that the process *P_1* has the type on the left hand side. This notation means that $\mathcal{A}, \mathcal{B}, \mathcal{C}$

P_1,Q, P_2 has type $\mathcal{A}, \mathcal{B}, \mathcal{C}$, respectively; more intuitively, this means that processes P_1,Q, P_2 are executed on machines $\mathcal{A}, \mathcal{B}, \mathcal{C}$, respectively. The type $\mathcal{A}|(\mathcal{B}|\mathcal{C})$ means a distributed system which consists of $\mathcal{A}, \mathcal{B}, \mathcal{C}$. The types of Spice-calculus [5] are defined by the following grammar.

$$\mathcal{A} ::= \mathbf{a} :: \{x_1, \ldots, x_n\} \mid (\mathcal{A}|\mathcal{B})$$

The typing of the Spice-calculus contributes to the following:

- identifying the origins of computational costs and
- formalizing current memory usage during execution of processes.

The type $\mathbf{a}$ :: $\{x_1,\ldots,x_n\}$ means that in the single computer node **a**, memory cells $x_1,\ldots,x_n$ are occupied.

In order to track memory usage accurately, stricter restriction is imposed in the Spice-calculus than in Milner's original process calculus, the pi-calculus. For example, a typing rule [5]

$$\frac{\mathbf{a} :: \mathcal{V} \triangleright P \quad \mathcal{V} \supseteq fv(M) \quad \mathcal{V} \supseteq fv(N)}{\mathbf{a} :: \mathcal{V} \triangleright \mathrm{out}\, M\langle N\rangle; P}\ \textbf{TypeOut}$$

The first assumption of this rule forces the process P to be executed on the *single* computer node **a**. Consequently, a term

$$\mathrm{inp}\, n(x); (p \mid Q)$$

is not allowed in the Spice-calculus, since (P | Q)$ must be executed on a distributed system composed of multiple computer nodes.

The Spice-calculus is given operational semantics as a transition relation, in other words, small-step semantics. We can find the computational cost consumed in each transition step of the semantics. Moreover, we can identify not only the level of the computational cost but also the origin of this cost. For example, [5], a rule

$$\frac{M \downarrow V : c}{\mathrm{hash}(M) \downarrow hash_V : c + hash}$$

formulate hash-value computation. The cost c is incurred for evaluation of M and *hash* for the hash-value computation. There are two kinds of transition relation between processes giving the operational semantics of the Spice-calculus: one is a reduction relation and another commitment relation. *Inner-process computation* is formalized as the reduction relation and *inter-process computation* as the commitment relation. The following is one of the rules defining the reduction relation:

$$\frac{M \downarrow V : c \quad N \downarrow V : d}{(\mathrm{match}\, M\, \mathrm{is}\, N\, \mathrm{err}\, \{P\}; Q) \ > \ Q : \mathbf{a} \cdot (c + d + match)}\ \textbf{RedMatch}$$

The costs for evaluating M and N are c and d, respectively. The resulting value of M and N is V. The following is one of the rules defining the commitment relation:

$$\frac{}{\mathbf{a} :: \mathcal{V} \vdash \ \mathrm{inp}\, n(x); P \xrightarrow{n} (x)P : \{\mathbf{a} \cdot store_x\}}\ \textbf{CommIn}$$

In this rule, the process inp $n(x)$; P is transit to an intermediate form (x)P, which is waiting for arrival of a message at the port n. The cost $store_x$ occurs at the node **a**.

Recently, Cervesato proposed another formal approach to quantitative analysis of secure protocols [8].

4.2.2 Formalization of Process Creation

In the existing version of the Spice-calculus [5], the costs of memory and computation, such as hashing and arithmetic operations, are formalized. However, the cost of process creation is overlooked. In this paper, we improve the Spice-calculus by adding the costs of process creation.

Actually, process creation entails a certain amount of cost. For example, in a variation of the Unix system, a Process Control Block (PCB) is allocated in a kernel memory space to save the process context when the process is created. Hence process-creating is considered as expensive. In order to formulate process creation in the Spice-calculus, we introduce several new rules for the operational semantics. In the reduction relation between processes, the process replication is represented as a rule

$$\text{repeat}\, P > P \mid (\text{repeat}\, P)$$

The cost of creating a process is represented as *process* and is incorporated into the rule as follows.

$$\text{repeat}\, P > P \mid (\text{repeat}\, P) : \mathbf{a} \cdot \mathit{process}$$

The terminated process is formalized as the nil stop. In order to formalize elimination of the terminated processes precisely, we introduce another kind of terminated process end and we add a new reduction rule in which stop is transit to end as follows.

$$\text{stop} > \text{end} : \mathbf{a} \cdot -\mathit{process}$$

When a process is terminated and eliminated, a cost *process* is deducted.

4.3 Example of Formalization: Three-Way Handshake

TCP's three-way handshake is described as follows in the Alice-and-Bob notation.

$$\begin{aligned} &A \to B : A, B, S_A \\ &B \to A : B, A, S_B, S_A + 1 \\ &A \to B : A, B, S_A + 1, S_B + 1 \end{aligned}$$

In the Spice-calculus, we can write the protocol as the following process expressions.

$$\begin{aligned} P_A \overset{def}{=} \ & \text{new}(S_A); \\ & \text{store}\, x_{sa} = S_A; \\ & \text{out}\, c\langle (A,\, B,\, x_{sa}) \rangle; \\ & \text{inp}\, c\, (p); \\ & \text{split}\, [x'_b,\, x'_a,\, x'_{sb},\, x'_{sa+1}]\ \text{is}\, p\ \text{err}\{\text{free}\, x_{sa},\, p\}; \\ & \text{free}\, p; \\ & \text{match}\, x'_{sa}\ \text{is}\, A\ \text{and}\, x'_{sb}\ \text{is}\, B\ \text{and}\, x'_{sa+1}\ \text{is}\ \text{succ}(x_{sa}) \\ & \quad \text{err}\big\{\text{free}\, x_{sa},\, x'_b,\, x'_a,\, x'_{sb},\, x'_{sa+1}\big\}; \\ & \text{free}\, x'_b,\, x'_a,\, x'_{sa+1}; \\ & \text{out}\, c\, \big\langle \big(A,\, B\, \text{succ}\, (x_{sa}),\ \text{succ}\, (x'_{sb})\big) \big\rangle; \\ & P'_A \end{aligned}$$

$$\begin{aligned}
P_B \overset{def}{=}\ & \text{new}(S_B);\\
& \text{inp}\, c\,(q_1);\\
& \text{split}\,[y_a,\, y_b,\, y_{sa}]\ \text{is}\, q_1\ \text{err}\{\text{free}\, q_1\};\\
& \text{free}\, q_1;\\
& \text{match}\, y_b\ \text{is}\, B\ \text{err}\{\text{free}\, y_a,\, y_b,\, y_{sa}\};\\
& \text{free}\, y_b;\\
& \text{store}\, y_{sb} = S_B;\\
& \text{out}\, c\langle(B,\, y_a,\, y_{sb},\, \text{succ}(y_{sa}))\rangle;\\
& \text{inp}\, c\,(q_2);\\
& \text{split}\,[y'_b, y'_a,\, y'_{sa+1},\, y'_{sb+1}]\ \text{is}\, q_2\ \text{err}\{\text{free}\, y_a,\, y_{sa},\, y_{sb},\, q_2\};\\
& \text{free}\, q_2;\\
& \text{match}\, y'_b\ \text{is}\, B\ \text{and}\, y'_a\ \text{is}\, y_a\ \text{and}\, y'_{sa+1}\ \text{is}\, \text{succ}(y_a)\\
& \text{and}\, y'_{sb+1}\ \text{is succ}\ (y_{sb});\\
& \quad \text{err}\left\{\text{free}\, y_a,\, y_{sa},\, y_{sb},\, y'_b,\, y'_a,\, y'_{sa+1},\, y'_{sb+1}\right\};\\
& \text{free}\, y'_b,\, y'_a,\, y'_{sa+1},\, y'_{sb+1};\\
& P'_B
\end{aligned}$$

Then a normal situation of communication between these two processes is represented as *NormalConfig*:

$$NormalConfig \overset{def}{=} (\text{repeat}\, P_A \mid \text{repeat}\, P_B).$$

On the other hand, a situation of a SYN-flood attack [9] on the three-way handshake protocol is described in Alice-and-Bob notation as follows.

$$\begin{aligned}
I \rightarrow B &: I_i,\, B,\, S_I\\
B \rightarrow I &: B, A, S_B, S_{I_i} + 1\\
&(i = 1, 2, \ldots)
\end{aligned}$$

The intruder in this attack is written as a process P_I as follows.

$$P_I \overset{def}{=} \text{new}(i);\ \text{new}(s);\text{out}\, c\langle(i,\, B,\, s)\rangle;\text{stop}.$$

Since the intruder spoofs the senders' IP-addresses, the responding packets from the victim B are lost. In order to represent such lost packets, we introduce a process P_N which formulates the network:

$$P_N \overset{def}{=} \text{inp}\, c(r);\text{stop}.$$

The attacking situation is written as a process *AttackConfig*:

$$AttackConfig \overset{def}{=} (\text{repeat}\, p_I \mid \text{repeat}\, p_B \mid \text{repeat}\, p_N).$$

Then the process is typed as

$$AttackConfig : (\mathbf{i}|\mathbf{b}|\mathbf{n}) \triangleright (\text{repeat}\, p_I \mid \text{repeat}\, p_B \mid \text{repeat}\, p_N).$$

Reducing the process *AttackConfig*, we know the level of costs consumed during execution.

$$\begin{aligned}
& \mathit{AttackConfig} \\
\twoheadrightarrow\ & \mathit{new}(S_B); (\mathrm{stop} \mid \mathrm{repeat}\, P_A \mid (\mathrm{inp}\, c\, (q_2); \cdots) \\
& |\mathrm{repeat}\, P_B | \,\mathrm{stop} \mid \mathrm{repeat}\, P_N) \\
& : \{\mathbf{i} \cdot \mathit{proces}, \mathbf{b} \cdot \mathit{proces}, \mathbf{n} \cdot \mathit{proces}\} \\
& \ +\{\mathbf{i} \cdot \mathit{pair}, \mathbf{b} \cdot \mathit{store}\} \\
& \ +\{\mathbf{i} \cdot -\mathit{proces}, \mathrm{b} \cdot 2\mathit{store}, \mathbf{b} \cdot \mathit{match}\} \\
& \ +\{\mathbf{b} \cdot \mathit{pair}, \mathbf{n} \cdot \mathit{store}\} \\
& \ +\{\mathbf{n} \cdot -\mathit{proces}\} \\
& \ = \{\mathbf{i} \cdot \mathit{pair}, \mathbf{b} \cdot \mathit{proces}, \mathbf{b} \cdot \mathit{pair}, \\
& \quad \mathbf{b} \cdot 3\mathit{store}, \mathbf{b} \cdot \mathit{match}, \mathbf{n} \cdot \mathit{store}\}
\end{aligned}$$

Here we know that not only memory cost but also process creation is consumed by the SYN-flood attack.

4.4 Conclusion

In our previous works, the Spice-calculus could analyze computational costs such as arithmetic operations, hash computation, and message transmission. However, the cost of process creation was disregarded. In this paper, we improved the Spice-calculus through adding cost evaluation of process creation and elimination. We extended the syntax of the cost in the Spice-calculus and operational semantics of the Spice-calculus. We then described TCP's three-way handshake and the SYN-flood attack as examples of our new calculus.

As well as the work presented in this paper, we have several other subjects to study in future. One of the most important ones is formalizing and analyzing the SYN-cookie method against the SYN-flood attack and other DoS-resistant methods [10, 11].

Acknowledgments This work was supported by Grants-in-Aid for Scientific Research (C) (24500009).

References

1. Schuba CL, Krsul IV, Kuhn MG, Spafford EH, Sundaram A, Zamboni D (1997) Analysis of a denial of service attack on TCP. In: Proceedings of the 1997 IEEE symposium on security and privacy, pp 208–223. IEEE Computer Society, IEEE Computer Society Press
2. Millen JK (1993) A resource allocation model for denial of service protection. J Comput Secur 2(2/3):89–106
3. Meadows C (1999) A formal framework and evaluation method for network denial of service. In: Proceeding of the 12th IEEE computer security foundations workshop, pp 4–13

4. Meadows C (2001) A cost-based framework for analysis of denial of service networks. J Comput Secur 9(1/2):143–164
5. Tomioka D, Nishizaki S, Ikeda R (2004) A cost estimation calculus for analyzing the resistance to denial-of-service attack. In: Software security—theories and systems. Lecture Notes in Computer Science, vol 3233. Springer, New York, pp 25–44
6. Milner R, Parrow J, Walker D (1992) A culculus of mobile processes, part i and part ii. Inf Comput 100(1):1–77
7. Abadi M, Gordon AD (1997) A calculus for cryptographic protocols: the spi calculus. In: Fourth ACM conference on computer and communication security, pp 36–47. ACM Press, New York
8. Cervesato I (2006) Towards a notion of quantitative security analysis. In: Gollmann D, Massacci F, Yautsiukhin A (eds) Quality of protection: security measurements and metrics—QoP'05, pp 131–144. Springer advances in information security 23
9. TCP SYN Flooding and IP spoofing attacks (1996), CA-1996-21
10. Aura T, Nikander P (1997) Stateless connections. In: International conference on information and communications security ICICS'97. Lecture notes in computer science, vol 1334, pp 87–97. Springer, Berlin
11. Aura T, Nikander P, Leiwo J (2001) DOS-resistant authentication with client puzzles. In: Security protocols, 8th international workshop. Lecture notes in computer science, vol 2133, pp 170–177. Springer, Berlin