RenPar'9,
Lausanne. Suisse, 20–23 mai 1997.

Objets Protégés Cryptographiquement

Uwe G. Wilhelm, Xavier Défago [a]

[a]Laboratoire de Systèmes d'Exploitation
Département d'Informatique
École Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)
{wilhelm,defago}@lse.epfl.ch

Mots-clé: agents, ingénierie inverse, résistance au tripatouillage, sécurité, confidentialité

1. Introduction

Les technologies basées sur le code mobile, telles que *Java*[1][4] et *ActiveX*[2][6] ont graduellement pris une place énorme et constituent un domaine où les enjeux sont particulièrement importants actuellement. Néanmoins, bien qu'un nombre croissant d'applications soient adaptées au paradigme du code mobile, un certain nombre de dangers subsistent, liés à une adoption massive de cette technologie. Certains de ces dangers, tels qu'une applet Java ou un contrôle ActiveX causant des dégâts dans son environnement d'exécution, sont adressés avec beaucoup de ferveur en raison des enjeux économiques importants qui y sont relatifs (sécurité dans Java [3], architecture d'authentification d'ActiveX [6]). D'autres problèmes dont les enjeux économiques ne sont pas moins importants, sont largement ignorés pour l'instant car les structures actuelles n'y sont pas adaptées et parce qu'ils nécessitent une nouvelle approche.

Cet article présente une approche différente permettant de garantir l'intégrité de l'environnement d'exécution, ainsi que de protéger le code et les données d'un objet contre toute manipulation ou divulgation. De plus, ces garanties restent valables avant, pendant et après l'exécution de l'objet. Une approche similaire a été prise par Herzberg et Pinter dans un contexte différent [5] mais, comme ces développements ont été effectués il y a plus d'une dizaine d'années, certains problèmes ne se présentent plus de la même manière et le concept nécessite une adaptation au contexte actuel.

La suite de l'article est organisée de la manière suivante. La section 2 donne une brève introduction au problème considéré. La section 3 permet d'introduire quelques définitions et notations de base, utilisées pour décrire le protocole. La section 4 introduit ensuite le protocole d'objets protégés cryptographiquement (*cryptographically protected objects*, CryPO) en détail. Finalement, la section 5 conclut l'article et résume les principales contributions.

2. Présentation du problème

Un objet actif O consiste en du code et des données pouvant être envoyés au travers d'un canal de communication. L'objet reçu par un autre site peut alors être exécuté sur une machine virtuelle. L'une des caractéristiques principales d'un objet est d'être indépendant de la plateforme. Il est identifié au moyen d'un nom O_{name}.

Un fournisseur qui envoie un objet à quelqu'un n'a actuellement aucune garantie de confidentialité pour le code et les données de celui-ci. Un problème se pose car les informations véhiculées par l'objet sont des valeurs appartenant au fournisseur — il peut s'agir d'algorithmes propriétaires — et, sans protection, pourraient être utilisées dans d'autres contextes, sans permission. Il s'agit d'un problème concret dont nous proposons un élément de solution avec le protocole d'objets protégés cryptographiquement CryPO.

3. Définitions et notations

Notre protocole CryPO étant basé sur des aspects importants de crytographie et de résistance au tripatouillage, nous commençons par donner un bref aperçu de ces concepts.

[1] *Java* est une marque déposée par Sun Microsystem Ltd.
[2] *ActiveX* est une marque déposée par Microsoft Corp.

3.1. Cryptographie

Une description détaillée des notions liées à la cryptographie dépassent largement le cadre de cet article. Plusieurs ouvrages de référence traitent de ce sujet de manière exhaustive [8, 2] et nous allons donc nous contenter de présenter les notations nécessaires à la description de notre protocole.

Le protocole utilise le principe de cryptographie à clé publique (tel que RSA [7]) où un *principal* P possède une paire de clés (K_P, K_P^{-1}). K_P est la clé publique de P, connue de tout le monde et K_P^{-1} est la clé privée uniquement connue de P. À l'aide de ces clés, il est possible de coder un message m, noté $\{m\}_{K_P} = m'$ que seul P peut lire avec sa clé privée; $\{m'\}_{K_P^{-1}} = m$.

Inversement, la plupart des systèmes à clé publique (notamment RSA) permettent de signer des messages. Un message m signé par P se note $\{m\}_{K_P^{-1}} = m_{sig}$. La signature peut être vérifiée par n'importe qui, au moyen de K_P: $\{m_{sig}\}_{K_P} = m$.

3.2. Résistance au tripatouillage

Le qualificatif de résistant au tripatouillage[3] est généralement utilisé pour désigner un module bien spécifié qui exécute une certaine tâche de type boîte noire. L'environnement extérieur ne peut pas interférer avec la tâche du module autrement que par le biais d'une interface très restreinte. Bien qu'il soit extrêmement difficile de construire un module complètement résistant au tripatouillage, de nombreux systèmes utilisent cette approche (p.ex., téléphones publics, cartes de crédit). Si suffisamment de temps et de ressources sont à disposition, la possibilité de violer les protections d'un TPE devient importante [1]. Par conséquent, le coût nécessaire[4], en temps et en ressources, pour casser les protections d'un environnement résistant au tripatouillage devrait être plus élevé que celui des informations qu'il doit protéger. Ainsi, pour la suite de cet article, nous faisons explicitement l'hypothèse que le coût nécessaire à percer les protections de notre TPE est beaucoup plus élevé que le gain pouvant résulter d'une telle action.

4. L'approche générale

Dans cette section, nous présentons le protocole permettant de garantir l'intégrité de l'environnement d'exécution et qui nous permet de protéger le code et les données des objets qu'il exécute. Nous commençons par présenter l'environnement d'exécution nécessaire, puis nous enchaînons sur la description du protocole qui l'utilise. La Figure 1 donne une représentation générale de l'architecture du système.

4.1. L'environnement d'exécution

Pour atteindre les objectifs fixés, nous avons besoin d'un environnement d'exécution particulier basé sur le concept d'environnement résistant au tripatouillage (*tamperproof environment*, TPE). La propriété principale est d'être un environnement d'exécution complet ne pouvant pas être inspecté ou modifié.

Un tel environnement doit être un ordinateur complet disposant d'un processeur, de mémoire vive et morte, ainsi qu'une unité de stockage stable (p.ex. disque dur, flash RAM). Cet environnement doit aussi posséder une machine virtuelle servant d'environnement d'exécution pour les objets et un système d'exploitation dont le rôle est de fournir une interface bien spécifiée pour accéder de l'extérieur au TPE et de contrôler la machine virtuelle (protection des objets les uns envers les autres). De plus, le TPE contient une clé privée connue d'aucune autre entité, qui reste inaccessible même à son propriétaire légitime. (voir 4.2).

Le TPE est connecté physiquement à un ordinateur hôte qui est sous le contrôle du propriétaire du TPE. Cet ordinateur hôte ne peut accéder aux fonctionnalités du TPE qu'au moyen d'une interface spécifique et clairement définie permettant notamment les opérations suivantes:

- téléchargement, migration et suppression d'objets,
- interactions hôte $\leftrightarrow$ objet et objet $\leftrightarrow$ objet, et
- vérification de certaines propriétés du TPE (telle qu'une liste des objects en cours d'exécution).

En raison de son implémentation en tant que module résistant au tripatouillage et de l'accès limité fournit par le système d'exploitation, il est impossible d'accéder directement à l'information que contient le TPE.

[3] En anglais «*tamperproof*». La traduction française a été reprise de [2].

[4] Il ne peut s'agir que d'une estimation et, de plus, ce coût a de fortes chances de diminuer avec le temps.

Toutes ces propriétés sont assurées et garanties par le constructeur du TPE[5] (*TPE manufacturer*, TM) qui fournit à l'utilisateur (*objet user*, OU) un certificat signé par le constructeur contenant des informations sur le type de TPE, ainsi que la clé publique de ce dernier. Le fournisseur de l'objet (*object provider*, OP) doit alors faire confiance au constructeur, qui lui assure que le TPE offre le degré de protection souhaité.

4.2. Génération des clés

Comme nous pouvons le supposer, il est capital que la clé privée du TPE reste secrète. Pour cela, le TPE doit générer lui-même sa clé privée et sa clé publique en se basant sur des données aléatoires prises de manière externe; p.ex. en échantillonant un bruit blanc. Ainsi, la clé n'est jamais directement disponible à l'exterieur du TPE et l'initialisation est effectuée chez le constructeur du TPE. D'autres approches permettant d'assurer la protection de cette clé, sont réalisables.

4.3. Protocole d'objets protégés cryptographiquement

Le protocole d'objets protégés cryptographiquement consiste à véhiculer des objets exclusivement de manière cryptée, de manière à ce qu'il soit impossible d'en analyser le contenu. Ce protocole peut facilement, et de manière transparente, être étendu pour empêcher toute modification du contenu en ajoutant une information qui garantit l'intégrité mais, ceci dépasse le cadre de cet article.

Le protocole est divisé en deux phases distinctes. La première consiste en une phase d'initialisation, exécutée une seule fois, avant l'exécution du protocole proprement dit. La seconde phase est celle qui concerne l'utilisation du TPE. Le protocole se base sur l'architecture illustrée dans la Figure 1.

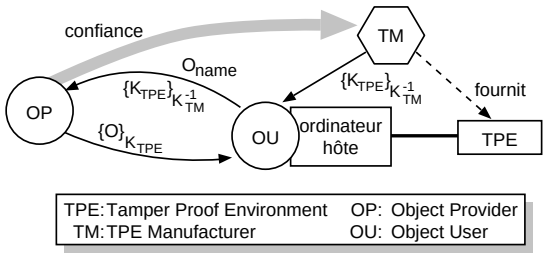

Figure 1: Architecture générale du système.

4.3.1. Initialisation

Dans la phase d'initialisation, les participants établissent les relations de confiance associées à chacune des différentes clés:

1. Le constructeur TM publie sa clé de certification K_{TM}.
2. Le constructeur TM envoie le certificat $Cert_{\mathrm{TPE}} = \{K_{\mathrm{TPE}}\}_{K_{\mathrm{TM}}^{-1}}$ à l'utilisateur.

4.3.2. Utilisation

Après que les participants aient obtenu les informations nécessaires, ils s'engagent dans la partie principale du protocole:

1. L'utilisateur OU envoie une requête au fournisseur OP avec le nom de l'objet qu'il souhaite obtenir O_{name}, et le certificat de son TPE: $Cert_{\mathrm{TPE}}$.
2. Le fournisseur OP analyse le certificat $Cert_{\mathrm{TPE}}$ et vérifie le type et le constructeur du TPE certifié, pour être sûr qu'il satisfait aux exigences de sécurité et de protection nécessaires. Il peut aussi vérifier d'autres informations liées à l'interaction, telle que l'identité de l'utilisateur, ou un éventuel paiement. Si la phase de vérification est passée avec succès, le protocole peut continuer.
3. Le fournisseur OP envoie à l'utilisateur OU l'objet crypté au moyen de la clé publique du TPE: $\{O\}_{K_{\mathrm{TPE}}}$.
4. L'utilisateur OU ne pouvant pas accéder aux informations contenues dans l'objet crypté, il ne peut rien faire d'autre avec $\{O\}_{K_{\mathrm{TPE}}}$ que de le télécharger dans le TPE.
5. Le TPE décode $\{O\}_{K_{\mathrm{TPE}}}$ au moyen de sa clé privée K_{TPE}^{-1}, ce qui lui permet d'obtenir l'objet O sous sa forme exécutable. En fonction des directives que l'utilisateur OU aura données au TPE lors du chargement de l'objet, ce dernier sera exécuté sur le TPE. Il pourra alors communiquer avec l'environnement local et pourra interagir avec d'autres objets ou même des applications distantes.

[5] Il serait envisageable de séparer les rôles du constructeur et du certifieur en deux organisations distinctes.

4.4. Utilité du protocole

Le protocole décrit ci-dessus garantit l'intégrité de l'environnement d'exécution et protège les données et le code d'un objet contre les modifications, manipulations et la divulgation. La garantie d'intégrité est basée sur la relation de confiance qui existe entre le fournisseur OP de l'objet et le constructeur TM du TPE. Dans la mesure où le fournisseur peut faire confiance au constructeur, et par conséquent au TPE certifié, il reste assuré que n'importe quel objet qu'il enverra à ce TPE sera protégé tant au niveau du code que des données car ceux-ci restent confidentiels. Ceci reste valable avant, pendant et après l'exécution de l'objet et sans aucun contrôle physique de l'environnement d'exécution.

Pour l'utilisateur OU de l'objet, il obtient la garantie d'un comportement confiné au domaine du TPE et que son ordinateur hôte n'aura à subir aucun dommage. Ceci est dû au confinement qu'offre le TPE et peut être augmenté par un protocole de certification des objets, similaire à ceux qui sont proposés pour Java ou ActiveX [3, 6].

En ce qui concerne le constructeur TM du TPE, il doit s'assurer que le TPE remplit toutes les spécifications, spécialement celles qui sont liées à la sécurité puisque tout le concept en dépend.

L'interface permet aux objets résidant sur le TPE de recevoir des messages. De plus, ces messages peuvent être cryptés au moyen de la clé publique du TPE pour que d'autres entités ne puissent pas avoir accès aux données contenues dans le message, pas même le propriétaire légitime du TPE. Un message peut être adressé à n'importe quel objet situé sur le TPE sans que personne ne puisse déterminer de quel objet il s'agit. L'anonymat du receveur du message reste ainsi préservé.

4.5. Les points faibles du protocole

Le protocole a des points faibles auxquels il s'agit de rester attentifs. Ceci concerne évidemment la clé secrète du TPE qui ne doit jamais pouvoir être accessible à quiconque. Dans le cas contraire, cela permettrait à quelqu'un de construire un faux TPE permettant de violer les garanties de sécurité et de confidentialité normalement offertes.

De la même manière et bien que les conséquences en soient un peu moins graves, les données contenues sur le TPE ne devraient jamais être accessibles autrement que par le fonctionnement normal de ce dernier.

5. Conclusion

Dans cet article, nous avons présenté un environnement d'exécution et un protocole permettant d'assurer la protection des objets qu'il exécute. Il permet d'empêcher l'ingénierie inverse du code de l'objet et de protéger le code et les données de celui-ci contre les manipulations et la divulgation.

Cet environnement d'exécution pourrait être mis-en-œuvre au moyen des technologies actuelles — p.ex., avec des cartes PC de type PCMCIA — sans que toutes les propriétés de résistance au tripatouillage ne soient présentes. Des recherches supplémentaires dans ce domaine sont encore nécessaires.

Une variation du protocole présenté peut sans doute être utilisée dans le cadre de la protection de données et de la confidentialité. Nous allons explorer cette approche de manière plus approfondie dans notre travail futur.

Bibliographie

1. R. Anderson and M. Kuhn. Tamper resistance — a cautionary note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, Oakland, California, November 1996. USENIX Association. `http://www.ft.uni-erlangen.de/ mskuhn/tamper.html`.
2. G. Brassard. *Cryptologie Contemporaine*. Masson, 1992.
3. J. S. Fritzinger and M. Mueller. Java security. White paper, Sun Microsystems, Inc., 1996. `http://www.javasoft.com/security/whitepaper.ps`.
4. J. Gosling and H. McGilton. The java language environment. White paper, Sun Microsystems, Inc., 1996. `http://www.javasoft.com/doc/language_environment/`.
5. A. Herzberg and S. S. Pinter. Public protection of software. In *Advances in Cryptology: CRYPTO'85*, pages 158–179, Santa Barbara, California, August 1985.
6. P. Johns. Signing and marking activex controls. *Developer Network News*, November 1996. `http://www.microsoft.com/intdev/security`.
7. RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard*. RSA Data Security, Inc., November 1993.
8. B. Schneier. *Cryptographie Appliquée*. International Thomson Publishing France, 1995.