Caso de éxito de método que aplica patrones de seguridad en la Ingeniería en Computación

Miguel Solinas[1], Jairo Trad[1], Juan Abdala[1], Francisco Capdevila[1], Eduardo B. Fernandez[2], Leandro Antonelli[3]

[1] Lab.de Arquitectura de Computadoras, FCEFN, UNC,
Av.Velez Sarsfield 1611, 5000 Córdoba Argentina.
msolinas@efn.uncor.edu, {jairotrad, tonisgo, fjcapdevila}@gmail.com,

[2] Dept.of Computer Science and Engineering
Florida Atlantic University, Boca Raton, FL, USA
ed@cse.fau.edu

[3] LIFIA; Fac.Informática; UNLP; Argentina;
leandro.antonelli@lifia.info.unlp.edu.ar

Resumen. Incorporar patrones de seguridad en etapas tempranas del proceso de desarrollo de sistemas de software es la forma más eficaz de construir sistemas seguros. Las preguntas que nos hacemos son uno: qué tan temprano es posible tenerlos en cuenta?; dos: qué método utilizar para minimizar el impacto de su incorporación?; tres: qué aporta la incorporación temprana de un patrón de seguridad al momento de la implementación.? En este trabajo presentamos un caso de éxito de un método para elicitar requerimientos de seguridad en etapas tempranas del proceso de desarrollo, mapearlos sobre patrones de seguridad y guiar las etapas posteriores del proceso de desarrollo de sistemas embebidos. Aplicando el método se construyó un "hardlock authenticator" utilizando un PIC, como práctica de laboratorio de la materia Sistemas de Computación de la carrera de Ing. en Computación de la Universidad Nacional de Córdoba.

Palabras Claves: Hardlock USB Authenticator, Security Pattern, Object Oriented Design, Software Engineering.

1 Introducción

Existen causas esenciales que producen vulnerabilidades y debilidades en los sistemas de software [7] y dichas causas se extienden a los sistemas embebidos. Una de ellas es la ausencia de herramientas y métodos adecuados para asistir las etapas tempranas del proceso de desarrollo de sistemas embebidos seguros.

En un sistema compuesto de hardware y software (habitual en Ingeniería en Computación) participan componentes de ambas categorías que interactúan con complejos sistemas de información. Por ello es de esperar que se presenten escenarios fuera del foco de los requerimientos funcionales que raramente son tenidos en cuenta al momento del análisis y diseño. Estos escenarios deben ser analizados a fin de detectar y prevenir debilidades en el producto final y sería deseable que puedan

mapearse sobre soluciones de seguridad conocidas a través de un mecanismo que facilite su estudio y análisis.

Por otro lado, si tenemos en cuenta 1) que la calidad y costo de producción de este tipo de sistemas depende fuertemente del conocimiento que tengamos de sus requerimientos; y 2) que los patrones de seguridad representan las mejores prácticas logradas por la industria a fin de detener o limitar ataques; concluimos que dos buenos criterios a incorporar en la producción de sistemas embebidos seguros son **uno:** promover la construcción de modelos de análisis y diseño que incorporen patrones de seguridad y **dos:** que estos modelos guíen las etapas posteriores del proceso de producción de sistemas embebidos.

Para comprender, estudiar y modelar el dominio de la seguridad descubrimos que LEL y Escenarios son herramientas adecuadas. Leonardi [6] y Antonelli [1] mostraron la eficacia de su uso en el estudio de otros dominios. LEL y escenarios se utilizan para capturar y abstraer conocimiento y comportamiento respectivamente del dominio. A partir de ellos se obtienen en forma sistemática tarjetas CRC [2]. Cuando los requerimientos del dominio cambian, LEL y los escenarios cambian y es posible mantener la consistencia con el problema que modelan. Luego, las tarjetas CRC también se modificarán. Estos cambios impactan en las siguientes etapas del ciclo de vida del sistema. Utilizamos LEL, Escenario y CRC para modelar un patrón de seguridad mediante un sub-escenario y conducir el estudio y análisis de problemas que demandan servicios de seguridad. El patrón de seguridad queda así identificado como una nueva entidad en el modelo: un sub-escenario.

Baseline Mentor Workbench ("BMW") es una herramienta que tiene como función asistir al experto del dominio durante la fase de ingeniería de requerimientos; administra entradas de LEL, escenarios y tarjetas CRC y permite tener forward traceability de requerimientos de seguridad.

En este trabajo describimos la construcción de un Hardlock Authenticator utilizando los adelantos realizados sobre la definición de un método que permite incorporar patrones de seguridad en el dominio específico de una aplicación utilizando la relación entre LEL y Escenarios para abstraer el conocimiento del dominio de la seguridad y la herramienta BMW para deducir CRC: clases candidatas.

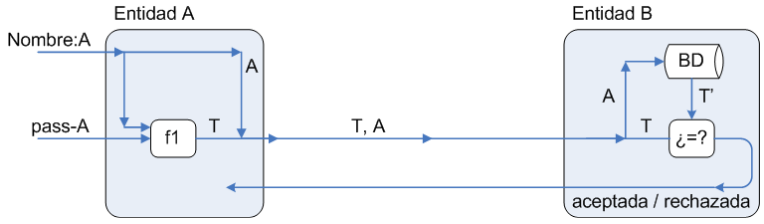
El trabajo está organizado de la siguiente manera: en 2 se presenta una breve descripción del problema de autenticación. En 3 se describe el patrón de seguridad Autenticación; se analiza el patrón y se muestran el LEL, Escenario y CRC encontrados utilizando BMW. En 4 se describe la aplicación con requerimientos de servicios de seguridad. Se muestran los resultados del análisis utilizando LEL, Escenarios y BMW. En 5 se describe brevemente el mecanismo propuesto. En 6 se expone paso a paso la implementación del diseño y en 7 se presentan algunas conclusiones de esta experiencia.

2 El problema de autenticación

En una red como Internet las entidades tienen presencia virtual y se reconocen por sus direcciones IP y mensajes que envían. Con ellos se identifican y suministran información o la solicitan. Esta presencia virtual conduce a graves riesgos de suplantación de identidad, lo que demanda servicios de seguridad que los limiten. El servicio más adecuado es la autenticación. Por otro lado, estudiar y analizar el escenario que demanda servicio de autenticación, es tan importante como identificar el servicio mismo ya que otros servicios como el de autorización y auditoría dependen de la autenticación. Es posible identificar dos tipos de autenticación, dependiendo del contexto en el que se provea este servicio: autenticación de entidad y autenticación de origen de datos. Nos interesa la Autenticación de entidad. Si bien las entidades en Internet, tienen presencia virtual, también tienen una existencia real con atributos y comportamientos que las identifican. Utilizar alguno de esos atributos y/o comportamientos, para identificarla virtualmente en una comunicación, es algo que resulta natural. De modo que también resulta natural, que el proceso de autenticación pueda construirse sobre 1) algo que la entidad posee; 2) algo que la entidad conoce; 3) algo que la entidad sabe hacer; ó 4) algún atributo físico. Nos interesan los casos 1) y 2). En la recomendación ITU-T X.509 "Athentication Framework" se definen dos esquemas básicos de autenticación: autenticación simple o débil y autenticación fuerte. Nos enfocaremos en la autenticación simple.

En la Figura 1 se muestra el esquema de autenticación simple que utilizamos, donde la entidad A, trata de probar su identidad frente a B, enviando su nombre y un autenticador T.

Figura 1: Esquema de autenticación simple



3 Patrón de seguridad Autenticación

Se describen exclusivamente aquellos títulos de la plantilla del patrón que son de interés para el trabajo.

3.2 Fuerzas

- Diferentes usuarios que pueden requerir más de una forma de autenticación. Necesitamos manejar esta variedad de usuarios.

- Necesitamos autenticar usuarios de forma confiable. Esto es un protocolo robusto y una forma de proteger los resultados de la autenticación.
- Hay proporcionalidad entre seguridad y costos. Sistemas más seguros usualmente som más costosos.

3.3 Solución

Usar un único punto de acceso para recibir las interacciones del sujeto con el sistema y aplicar un protocolo para verificar la identidad del sujeto. El protocolo utilizado puede implicar que el usuario ingrese algún valor conocido o puede ser un proceso mas elaborado. En la Figura 2 se muestra el diagrama de clases para el patrón. Un "Subject", típicamente es un usuario, solicita acceso a los recursos de un sistema. "Authenticator" recibe esta petición y aplica un protocolo utilizando alguna "Authentication Information". Si la autenticación es exitosa, "Authenticator" crea una "Proof of Identity" (esto puede ser explícito, por ejemplo un Token, o implícito). En la Figura 3 se muestra la dinámica del proceso de autenticación. Al usuario se le retorna un "Handle" como prueba de identidad.

Figura 2: Diagrama de clases patrón autenticación

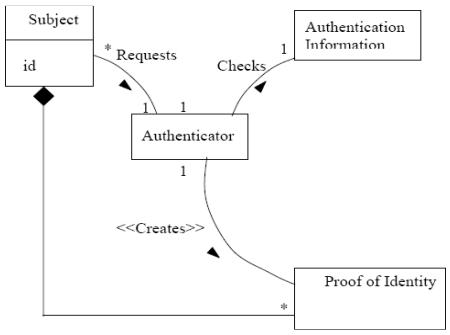
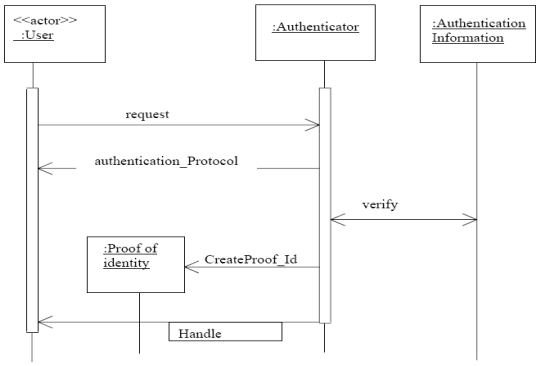


Figura 3: Dinámica de autenticación



3.4 Consecuencias

- Dependiendo del protocolo y la información de autenticación utilizada, podemos manejar cualquier tipo de usuarios y autenticarlos de diversas maneras.
- Dado que la información de autenticación es separada, podemos almacenarla en un área protegida, donde todos los sujetos puedan tener acceso de solo lectura.
- Podemos utilizar una variedad de algoritmos y protocolos de diferentes fortaleza para el proceso de autenticación. Depende de la relación entre seguridad y costos.
- Podemos producir una prueba de identidad para utilizar en reemplazo de autenticaciones posteriores. Esto mejora la performance del sistema.

3.5 Términos de LEL y Escenario identificados y CRC deducidas para patrón Autenticación

Las entradas de LEL para el patrón Autenticador se muestran en la Tabla 1. El escenario identificado se muestra en la Tabla 2. Las CRC primarias y secundarias deducidas utilizando BMW en la Tabla 3.

Tabla 1: Entrada de LEL de patrón Autenticador

LEL #1	Sinónimo	Authenticator (sujeto)
	Noción	Medio utilizado para acreditar la identidad de User.
	Impacto	Separa UserId y TestigoUser contenido en Request Solicita Testigo de UserId a InformationAuthentication Crea ProofId
LEL #2	Sinónimo	InformationAuthentication (objeto)
	Noción	Entidad que conoce el Testigo de User
	Impacto	Entrega información solicitada por Authenticator.
LEL #3	Sinónimo	ProofId (objeto)
	Noción	Instrumento utilizado por Authenticator para acreditar la identidad de User.
	Impacto	User conoce su ProofId.
LEL #4	Sinónimo	Request (objeto)
	Noción	Medio utilizado por *User* para solicitar verificación de identidad a Authenticator.
	Impacto	Es solicitado por *Authenticator*. *Authenticator* usa información contenida en *Request* para consultar InformationAuthenticator.
LEL #5	Sinónimo	User (sujeto)
	Noción	Entidad que solicita verificar su identidad con *Authenticator*.
	Impacto	Envía *Request* a *Authenticator*. Espera aceptación o rechazo de *Request*.

Tabla 2: Escenario de patrón Autenticador

Título	Solicita autenticación a *Authenticator*
Objetivo	Verificar identidad de *User*
Contexto	Comunicación establecida entre *User* y *Authenticator*
Recurso	*Request* enviado por *User* a *Authenticator*
Actores	*User*, *Authenticator*
Episodios	*User* calcula TestigoUser. *User* crea *Request* y envía TestigoUser mas UserId a *Authenticator* *Authenticator* separa TestigoUser y UserId *Authenticator* solicita Testigo de UserId a *InformationAuthentication*

	Authenticator recibe Testigo de UserId *Athenticator* compara TestigoUser y Testigo Si TestigoUser es igual a Testigo, *Authenticator* crea *ProofId* y comunica decisión a *User*

Tabla 3: CRC deducidas

CRC #1	Primary CRC Card	*Authenticator*
	Responsabilities	Separa UserId y TestigoUser contenido en *Request* Solicita Testigo de UserId a *InformationAuthenticator* Crea *ProofId*
	Collaborations	*Request*, *User*, *InformationAuthenticator*, *ProofId*
CRC #2	Primary CRC Card	*User*
	Responsabilities	Envía *Request* a *Authenticator* Espera aceptación o rechazo de *Request*
	Collaborations	*Request*, *Authenticator*, *InformationAuthenticator*, *ProofId*
CRC #3	Secondary CRC Card	*InformationAuthenticator*
	Responsabilities	Entrega información solicitada por *Authenticator*
	Collaborations	*Request*, *User*, *Authenticator*, *ProofId*
CRC #4	Secondary CRC Card	*ProofId*
	Responsabilities	User conoce su ProofId
	Collaborations	*Request*, *User*, *Authenticator*, *InformationAuthenticator*
CRC #5	Secondary CRC Card	*Request*
	Responsabilities	Es solicitado por *Authenticator* Authenticator usa información contenida en Request para consultar InformationAuthenticator
	Collaborations	*User*, *Authenticator*, *InformationAuthenticator*, *ProofId*

4 Descripción de la aplicación

Diseñar y construir un mecanismo de autenticación para Linux que utilice como elemento controlador del proceso de autenticación un hardlock USB. Una vez autenticado, el usuario debe tener acceso a la aplicación anfitriona. Si el hardlock se retira, la aplicación anfitriona se debe cerrar inmediatamente. Una vez autenticado el usuario, se podrá ejecutar una sola instancia de la aplicación anfitriona.
Se obtienen dos casos de uso que se modelan como escenarios: Login de usuario y Retirar Hardlock.

4.1 Términos de LEL, Escenario y CRC deducidas para aplicación

Las entradas de LEL para la aplicación se muestran en la Tabla 4. Los escenarios identificados se muestran en la Tabla 5 y 6. Las CRC primarias y secundarias deducidas utilizando BMW en la Tabla 7.

Tabla 4: Entrada de LEL para aplicación

LEL #1	Sinónimo	Authenticator (sujeto)
	Noción	Medio utilizado para acreditar la identidad de Usuario.
	Impacto	Separa UserId y TestigoUser contenido en Request

		Solicita Testigo de UserId a Hardlock
LEL #2	Sinónimo	Hardlock / InformationAuthenticator (objeto)
	Noción	Entidad que conoce Testigo de Usuario
	Impacto	Entrega información solicitada por Authenticator.
LEL #3	Sinónimo	Intérprete (sujeto)
	Noción	Intermediario entre User y Authenticator Entidad que solicita verificar la identidad de User.
	Impacto	Envía Request a Authenticator Espera aceptación o rechazo de *Request*

Tabla 5: Escenario "Login de usuario"

Título	Login de usuario
Objetivo	Acceder a los servicios de aplicación anfitriona.
Contexto	PC encendida, Hardlock conectado a puerto USB
Recurso	Nombre de usuario, clave
Actores	Usuario, Interprete
Episodios	Usuario ingresa UserId y clave Sistema operativo ejecuta Intérprete asociado a UserId Intérprete ***solicita autenticación a Authenticator***

Tabla 6: Escenario "Retirar Hardlock"

Título	Retirar Hardlock.
Objetivo	Detener ejecución de aplicación
Contexto	Usuario autenticado exitosamente
Recurso	PC encendida
Actores	Usuario, Hardlock
Episodios	Usuario retira Hardlock de puerto USB Sistema operativo detecta retiro de dispositivo Sistema operativo detiene ejecución de aplicación anfitriona.

Tabla 7: CRC deducidas para aplicación

CRC #1	Primary CRC Card	*Hardlock*
	Responsabilities	Entrega información solicitada por *Authenticator*
	Collaborations	*Authenticator*
CRC #2	Primary CRC Card	*Intérprete*
	Responsabilities	Envía Request a Authenticator Espera aceptación o rechazo de Request
	Collaborations	*Authenticator*
CRC #3	Secondary CRC Card	*Authenticator*
	Responsabilities	Separa UserId y TestigoUser contenido en Request Solicita Testigo de UserId a Hardlock
	Collaborations	*Intérprete*

El diagrama de objetos de la realidad de la aplicación, construido a partir de las CRC deducidas para el patrón de seguridad se muestra en la Figura 6. Se observar que los objetos "User" e "InformationAuthentication" del Patrón Autenticación han sido reemplazados por los objetos Intérprete y Hardlock respectivamente. En la Figura 7 se muestra el diagrama de secuencia, en donde a ProofId se le ha asignado la responsabilidad de chequear permanentemente la presencia del hardlock en el puerto USB.

Figura 6: Diagrama de clases de aplicación

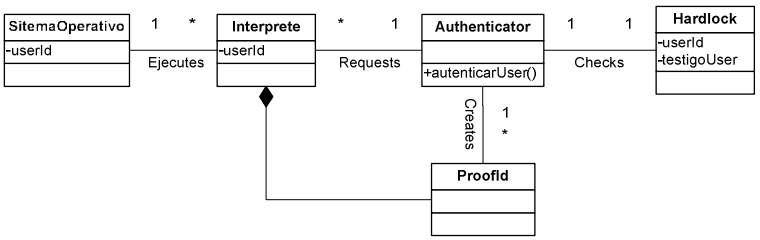
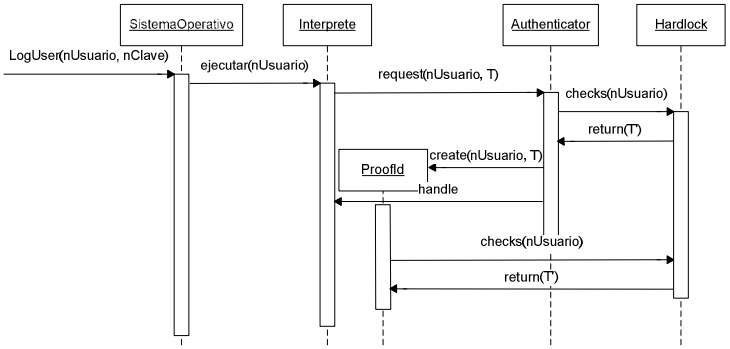


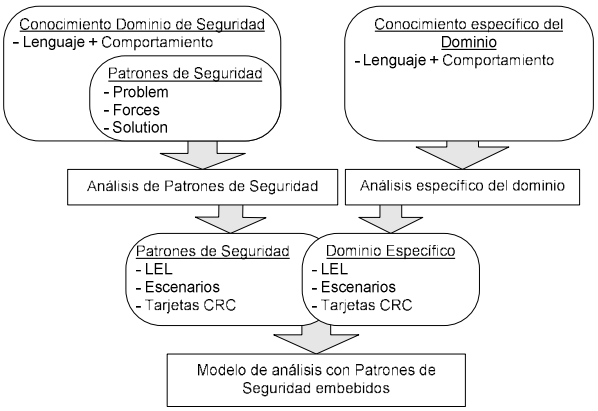
Figura 7: Diagrama de secuencias de aplicación



5 Mecanismo propuesto

El mecanismo para incorporar patrones de seguridad en etapas tempranas del proceso de desarrollo de un sistema de software se muestra en la Figura 7. Abstraemos el conocimiento y el comportamiento del dominio de la seguridad con LEL y Escenarios y utilizamos BMW para deducir CRC: clases candidatas. Asociamos un patrón de seguridad con un sub-escenario. Este modelo, junto a la especificación del patrón, es el que conduce el estudio y análisis del dominio con servicios de seguridad. Es importante poder modelar el patrón como un sub-escenario entre los requerimiento de seguridad, a fin de analizarlo con la misma herramienta e incorporarlo en el estudio y análisis de requerimientos.

Figura 8: Descripción del Mecanismo Propuesto



6 Implementación

Cuando un usuario intenta iniciar sesión en Linux, el sistema operativo solicita usuario y contraseña. Los datos ingresados se comparan con las tuplas: [usuario:contraseña(hash): intérprete_de_comandos] contenidas en (/etc/passwd). Cuando la comparación es exitoso, se ejecuta el intérprete de comandos asociado al par usuario:contraseña. Es un esquema de autenticación simple como el descripto en la Figura 1. La estrategia elegida para forzar la autenticación utilizando un hardlock conectado al puerto USB es reemplazar el intérprete de comandos por un ejecutable encargado de comprobar el hash con la información almacenada en el hardlock. Es necesario que este ejecutable tome el "username" actual y el "hash" asociado para el proceso de autenticación. Una vez que un usuario se ha autenticado con el sistema operativo es necesario instanciar un intérprete de comandos. Como el intérprete de comandos se ha reemplazado, se lanza un proceso hijo que llama a un intérprete de comandos. En este caso se instancia bash. Por un lado, el proceso padre queda en un bucle verificando que el hardlock siga en su lugar. Si durante la sesión se lo extrae el padre muestra un mensaje de error y finaliza su ejecución, dejando huérfano al proceso hijo. Luego, el sistema operativo se encarga de detectar y finalizar todos los procesos huérfanos. Por otro lado, el proceso padre debe detectar la finalización del proceso hijo. Cuando esto ocurra, él debe también, finalizar su ejecución.

6.1 Configuración de pic-usb.

Para implementar el hardlock se utiliza el microcontrolador PIC18F2455 de Microchip. Cumple con los requerimientos propuestos por la Cátedra: a) Disponibilidad; b) Bajo costo; c) Posibilidad de conexión a puerto USB en una arquitectura X86; d) Capacidad de funcionar con la alimentación que brinda el puerto USB y e) Conocimiento previo del microcontrolador.

Se evaluaron diferentes lenguajes y métodos para la configurar la interfaz USB del microcontrolador: a) Soluciones brindadas por el fabricante; b) Distintas variantes de C; c) Ensamblador y d) la solución bootloader. Se utilizó esta última, perteneciente al proyecto pingüino [11]: se trata de una versión primitiva de BIOS, cuya función es pre-cargar la configuración básica del dispositivo USB. Pingüino es un Kit de desarrollo basado en Arduino [12], incluye una IDE multiplataforma con el compilador C, SDCC; el ensamblador para productos Microchip; un linker (GPUTILS) y un bootloader basado en el proyecto VASCO PUF. Todos estos componentes se encuentran bajo licencias libres (LGPL o GPL según corresponda). Por otro lado, el kit contiene toda la información para el ensamblado del circuito. Ésta se encuentra publicada bajo licencia Open hardware.

Para la comunica **PC-PIC s**e utilizó la librería LibUSB v1.0 [13]. Se trata de un conjunto de rutinas que pueden ejecutarse en modo usuario para el control de la transmisión de datos desde o hacia dispositivos USB, sin la necesidad de utilizar drivers en modo kernel. Mediante las funciones disponibles se realiza la detección, conexión, comunicación y desconexión del dispositivo USB.

6.2 Modelo Estático y dinámico de la aplicación

En las Figuras 8 y 9 se muestran el diagrama de clases y diagrama de secuencia de la implementación.

Figura 8: Diagrama clases de la implementación

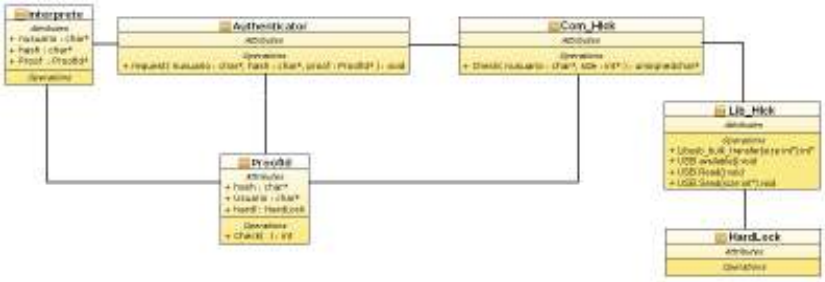
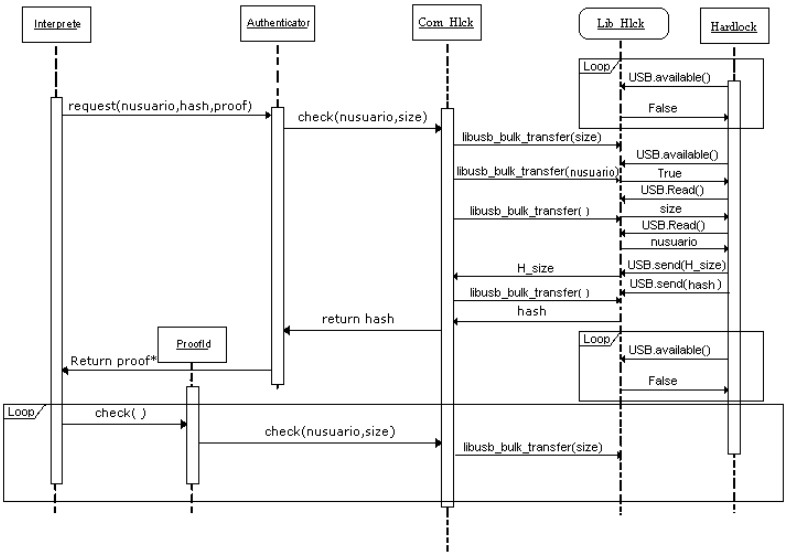


Figura 9: Diagrama de secuencia de la implementación



7 Conclusiones

El trabajo muestra que es posible incorporar patrones de seguridad en etapas muy tempranas del proceso de desarrollo de software. Con el mecanismo propuesto se pueden estudiar y analizar patrones de seguridad y descubrirlos en dominios específicos. Así presentado el conocimiento y el comportamiento de un patrón de seguridad hace que su estudio y posterior análisis tenga una curva de aprendizaje muy suave. Esto conduce rápidamente a un modelo de objetos de la aplicación que cuenta con el lenguaje y el comportamiento del patrón de seguridad, y a su incorporación en etapas muy tempranas del proceso de desarrollo.

Es posible tener trazabilidad de requerimientos de seguridad. Si los requerimientos de seguridad cambian, por cambios en las políticas de seguridad o en los requerimientos propios del dominio, es posible describir y seguir el impacto de estos cambios hasta la etapa de diseño.

Se han analizado dominios más complejos de aplicaciones reales que se encuentran en producción, con mayor número de elementos del LEL y Escenarios. Se ha observado que al aplicar el método con no expertos en el dominio de la seguridad, se hace muy útil disponer de un catálogo de patrones de seguridad expresados en términos de LEL, Escenarios y CRC, para facilitar su estudio, selección e incorporación como sub-escenarios en los episodios que describen comportamiento que requieren servicios de seguridad. De este modo, cada patrón de seguridad puede tener asociado, no sólo una UML “class model”, sino también un LEL, Escenario y

CRC “model”, que lo posiciona sobre la etapa de elicitación de requerimientos. Esto ayuda a una mejor comprensión, estudio y modelado del dominio de la seguridad.

E.Fernandez ha propuesto una aproximación diferente para aplicar los patrones de seguridad en un modelo conceptual [3]. Necesitamos comparar nuestras aproximaciones para ver si pueden complementarse entre ellas. Otra aproximación se basa en aspectos [4], también la compararemos con nuestra aproximación.

Referencias

1. Antonelli L.,“Traceability en la elicitación y especificación de requerimientos”; Facultad de Informática – Universidad Nacional de La Plata, Argentina, 2003.
2. http://en.wikipedia.org/wiki/Class-Responsibility-Collaboration_card
3. Fernandez, E.B. and Yuan, X.Y. " Securing analysis patterns", *Procs. of the 45th ACM Southeast Conference (ACMSE 2007)*, March 23-24, 2007, Winston-Salem, North Carolina.
4. Georg, G., Ray, I., Anastasakis, A., Bordbar, B., Toahchoodee, M., and Houmb, S.H., “An aspect-orinted methodology for designing secure applications”, Inf. And Software Technology, 51 (2009), 846-864.
5. Leite, J.C., Rossi, G., et al.: “Enhancing a Requirements Baseline with Scenarios”. Proceedings of RE 97’, IEEE Third International Requirements Engineering Symposium, IEEE Computer Society Press, 1997, 44-53.
6. Leonardi, C., Leite J.C., Rossi G., “Una Estrategia de Modelado Conceptual de Objetos basada en Modelos de Requisitos en Lenguaje Natural”, Tesis de maestría (M.S. Thesis), http://www-di.inf.puc-rio.br/~julio/teses.htm, Facultad de Informática, Universidad Nacional de La Plata, Argentina, Noviembre, 2001.
7. Schumacher M., Fernandez, E. B., Hybertson, D., Buschmann, F., and Sommerlad, P., Security Patterns: Integrating security and systems engineering", Wiley 2006.
8. Brown F.L., Fernandez E.B., "The Authenticator pattern", Procs. of Pattern Languages of Programs Conf. (PLoP99), http://jerry.cs.uiuc.edu/~plop/plop99
9. Hays V., Loutrel M., Fernandez E.B., "The Object Filter and Access Control Framework", Procs. of PLoP 2000, http://jerry.cs.uiuc.edu/~plop//plop2k/proceedings/proceedings.html
10. SAML, http://www.saml.org and http://www.oasis-open.org/committees/security/
11. Proyecto Pinguino; http://www.hackinglab.org/pinguino/index_pinguino.html
12. Arduino; http://www.arduino.cc/es/
13. LibUSB; http://libusb.sourceforge.net/api-1.0/