VULNERABILIDADES DE VALIDACIÓN DE ENTRADA

Explotación, Modelo de Ataques y Riesgo Residual

Juan Rafael Álvarez Correa – Fluidsignal Group

Juan Guillermo Lalinde Pulido – Universidad EAFIT

ORDEN DEL DÍA

- Definición del problema
- Solución propuesta
- Alcance
- Trabajo previo
- Desarrollo
- Conclusiones
- Trabajo futuro

PROBLEMA - CONTEXTO

- Incidentes de seguridad
  - **2002** - 82.094 (CERT)
  - **2003** - 137.529 (CERT)
- Vulnerabilidades
  - **2004** - 3053 (MITRE-CVE)
- Técnicas de explotación
  - **1996** – Smashing the stack for fun an profit, *AlephOne*
  - **1997** – How to write buffer overflows, *Mudge*
  - **1998** – The tao of windows buffer overflow, *DilDog*

- Técnicas de explotación
  - **1998** - Defeating solar designer's non-executable stack patch, *Wojtczuk*
  - **1999** – The frame pointer overwrite, *Klog*
  - **1999** – w00w00 on heap overflows, *Matt Conover*
  - **2000** – Bypassing Stackshield and Stackguard, *Bulba*
  - **2001** – Exploiting format strings vulnerabilities, *Scut*
  - **2001** – Once upon a free(), *Anonymous*
  - **2001** – Overwriting the .dtors section, *Bello Rivas*
  - **2002** – Four different tricks to bypass Stackshield and Stackguard protection, *Gerardo Richarte*
  - **2005** – How to hijack the Global Offset Table with pointers for root shells, *c0ntext*

PROBLEMA

- Los desarrolladores de software desconocen los *errores de programación* que generan riesgos, y los *controles existentes*
- Los desarrolladores de controles desconocen *todos los posibles ataques* y sólo mitigan parte del problema
- Los usuarios desconocen el *nivel de exposición* cuando utilizan todos los controles

PROBLEMA - IMPLICACIONES

- Costos y complejidad alta en la eliminación de problemas de seguridad
- Nuevas variedades de ataques a vulnerabilidades conocidas previamente. Éstas **pueden ser anticipadas**
- Controles sólo para tipos de ataques específicos y no para mitigar todo el riesgo
- Falsa sensación de seguridad de los usuarios
- Desconocimiento de la efectividad de mitigación de cada control

SOLUCIÓN PROPUESTA

- Entender el problema
  - Técnicas de explotación
- Caracterizar los posibles ataques
  - Modelo de ataques
- Identificar la efectividad de los controles
  - Análisis de riesgo (Riesgo residual)
- Verificar el modelo
  - Auditoría de código fuente
- Transferir conocimiento
  - Divulgación

- Vulnerabilidades de validación de entrada
  - *Suposiciones sobre la longitud de la entrada*
    - Desbordamiento de buffer
  - *Suposiciones sobre el rango de la entrada*
    - Desbordamiento de enteros
  - *Suposiciones sobre el contenido sintáctico de la entrada*
    - Inyección de HTML/JS/Comandos/SQL/Cadenas de formato
- Vulnerabilidades de sincronización
- Vulnerabilidades de configuración

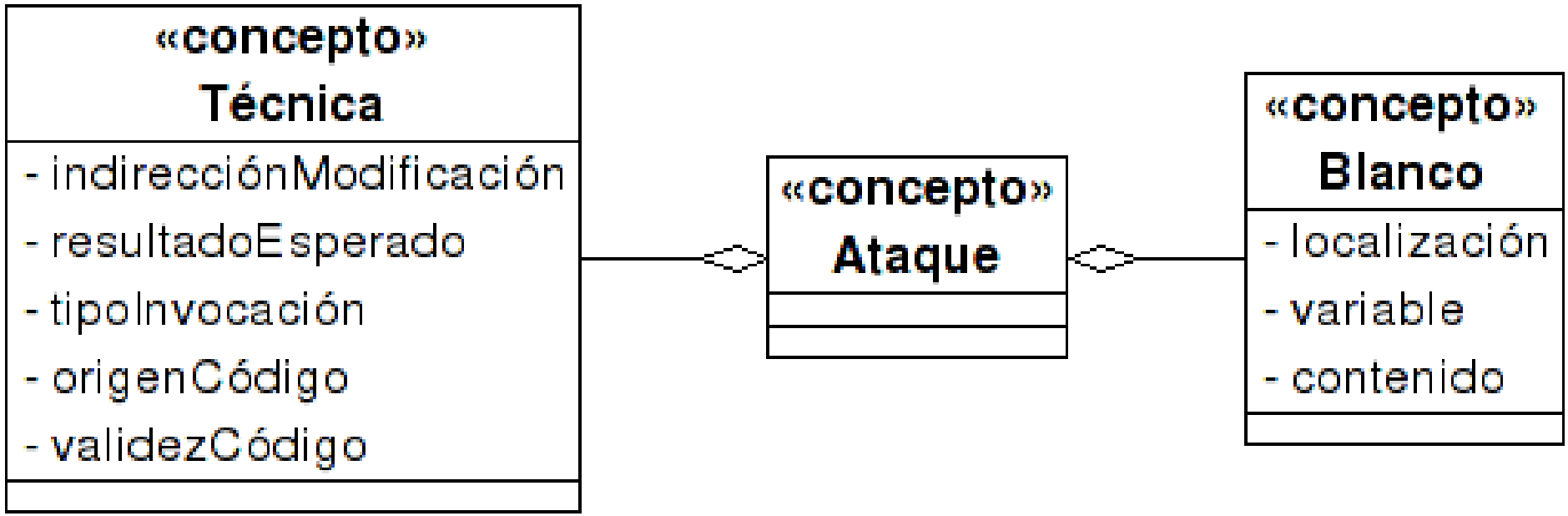
TRABAJO PREVIO

- A comparison of publicly available tools for dynamic buffer overflow prevention (Wilander, In Proceedings)
  - Enfocado a un solo tipo de vulnerabilidad
  - Enfocado a herramientas no a controles conceptuales
  - No se identifica el riesgo residual
- An overview of common programming security vulnerabilities and possible solutions (Younan, Master Thesis)
  - No se caracterizan todos los posibles ataques
  - Enfocado en herramientas, no en controles

- **Desbordamiento de buffer**
  - Explotación
  - Modelo de ataques
  - Análisis de riesgo
- **Desbordamiento de enteros**
  - Explotación
  - Modelo de ataques
  - Análisis de riesgo
- **Inyección de contenido sintáctico**
  - Explotación
  - Modelo de ataques
  - Análisis de riesgo

- **Explotación**
  - Explotación por sobreescritura de la dirección de retorno con código válido inyectado
  - Explotación por sobreescritura de la dirección de retorno con código válido no inyectado
  - Explotación por sobreescritura de la dirección del marco de pila con código válido inyectado

- **Modelo de ataques**
  - Universo total (160 combinaciones)

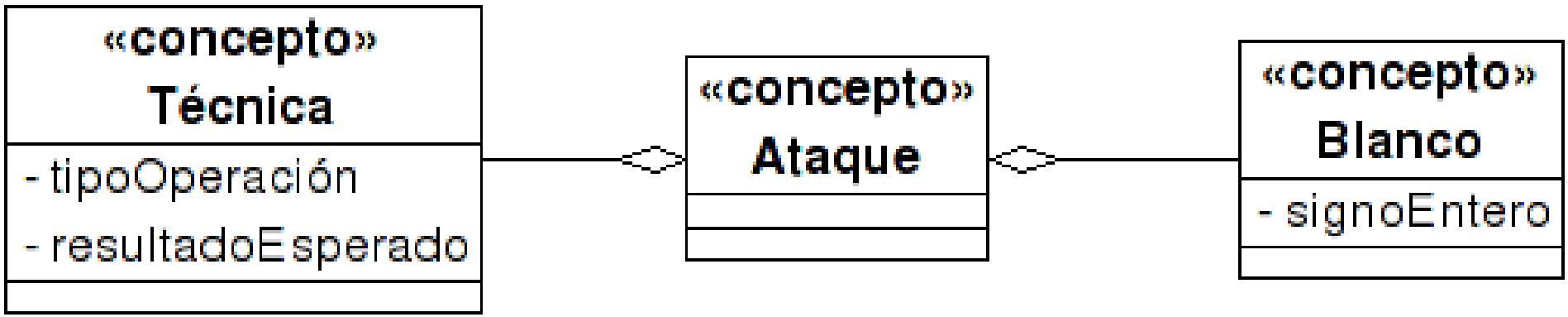


- **Análisis de riesgo - Observaciones**
  - Los controles con aplicabilidad en la fase de desarrollo de software (verificación de limites, redimensionamiento de arreglos) son los mas efectivos.
  - Los controles que aplican cuando el software está construido mitigan un pequeño porcentaje de ataques y con altos impactos (interrupción)
  - La mayoría de controles protegen los blancos en la pila, sin embargo sólo son un 50%
  - Ningún control protege el resultado cambio de información

- **Análisis de riesgo – Soluciones**
  - Extensiones de compilador que mantengan respaldo de las direcciones que cambian flujo. Objetivo: Continuar el flujo de la ejecución
  - Construir analizador sintáctico con análisis de flujo de datos. Objetivo: Disminuir el número de falsos positivos
  - Extensiones al compilador para canarios en otras regiones de memoria. Objetivo: Detectar el overflow en otras zonas de memoria.
  - Hash en el espacio del kernel para direcciones de memoria que permitan la transferencia de flujo

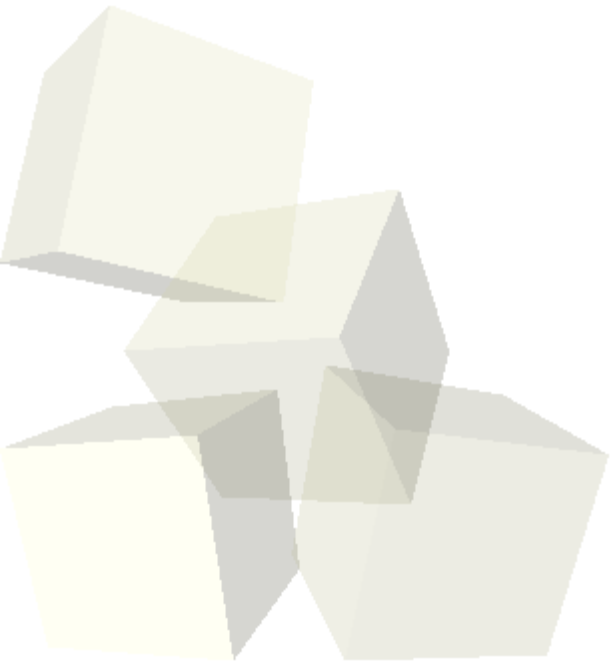
- **Explotación**
  - Explotación de entero con signo por coerción para volver a empezar
  - Explotación de entero con signo por coerción para cambiar de signo
  - Explotación de entero sin signo por coerción para volver a empezar
  - Explotación de entero con signo por aritmética para volver a empezar
  - Explotación de entero con signo por aritmética para cambiar de signo
  - Explotación de entero sin signo por aritmética para volver a empezar

- **Modelo de ataques**
  - Universo total (8 combinaciones)



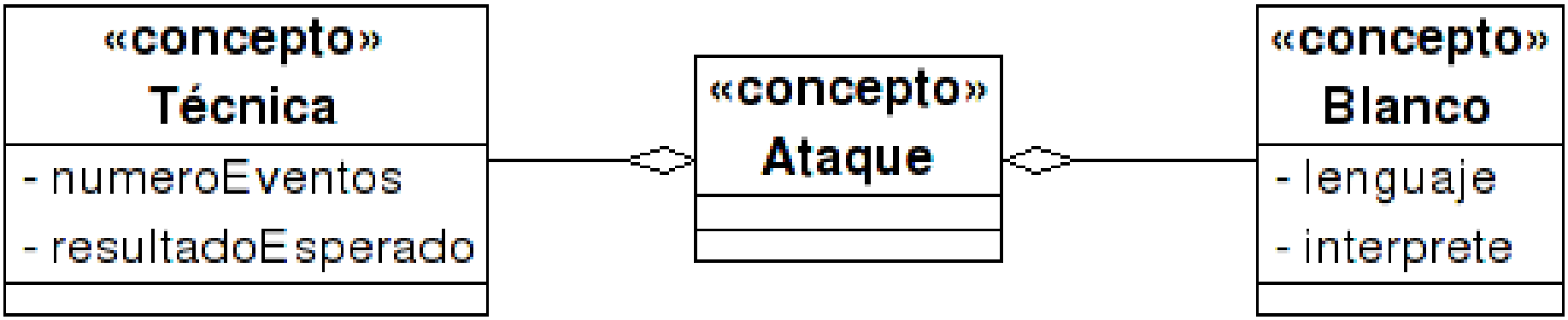
- **Análisis de riesgo - Observaciones**
  - El control más efectivo son las invariantes de tipo precondición. Este control no sólo los detecta sino que los previene
  - Las poscondiciones para algunas operaciones aritméticas no es posible de implementar matemáticamente y requiere soporte del hardware
  - Las poscondiciones pueden no detectar el desbordamiento con resultados de tipo volver a empezar

- **Análisis de riesgo – Soluciones**
  - Construir un programa de análisis sintáctico que permita evaluar los lugares donde se hace coerción de enteros como posibles lugares de desbordamiento
  - Construir traps para GCC con precondiciones y no con poscondiciones



- **Explotación**
  - Inyección de HTML de primer evento
  - Inyección de comandos de sistema operativo de primer evento
  - Inyección de SQL de primer evento
  - Inyección de JavaScript de segundo evento
  - Inyección de cadenas de formato de primer evento para leer posiciones arbitrarias de memoria
  - Inyección de cadenas de formato de primer evento para escribir posiciones arbitrarias de memoria
  - Inyección de cadenas de formato de primer evento para ejecutar código

- **Modelo de ataques**
  - Universo total (54 combinaciones)



- **Análisis de riesgo - Observaciones**
  - La aleatorización de sentencias disminuyen la probabilidad de todos los tipos de ataque, pero sólo existen implementaciones para SQL
  - Los cortafuegos de aplicación disminuyen la probabilidad de todos los tipos de ataques pero sólo existen para aplicaciones Web.
  - El control de escapado de caracteres y los API de serialización son los controles más efectivos
  - La verificación de contaminación en tiempo de compilación sólo aplica actualmente a C
  - Cuando un interprete soporta más lenguajes aumenta la probabilidad de ataque

Análisis de riesgo – Soluciones

- Diseñar un mecanismo universal de escapado de caracteres, de forma que se facilite la construcción de API de serialización genéricas
- Implementar cortafuegos de aplicación para aplicaciones de todo tipo
- Implementar análisis de contaminación en el lenguaje bash para la invocación de shells anidados
- Implementar APIs de serialización para Shell Script, LDAP filters y cadenas de formato
- Implementar estrategias de aleatorización de instrucciones para HTML, JS, etc.

- **Metodología**
  - Seleccionar aplicaciones a auditar
  - Seleccionar herramientas a utilizar
  - Obtener posibles vulnerabilidades
  - Identificar variables relacionadas con la posible vulnerabilidad
  - Seguir hacia atrás el flujo de datos
  - Identificar el orígen de la información
  - Verificar la existencia de la vulnerabilidad mediante explotación

- **Resultados**
  - Desbordamientos de buffer en:
    - Keepalived libipfwc.c linea 315
    - Keepalived libipfwc.c linea 395
    - SRG main.cc linea 1149
    - BIRD io.c linea 76
    - BIRD client.c linea 253
  - Condición de competencia
    - SRG utils.cc linea 495

- **Bogota – ACIS:** Jornadas Nacionales de Seguridad de la Información ACIS
- **Medellín – EAFIT:** Tertulia de Seguridad de la Información
  - Desbordamiento de buffer
  - Inyección de contenido sintáctico HTML/JS/SQL
  - Inyección de contenido sintáctico format strings
- **Valparaíso – UTFSM:** Congreso Iberoamericano de Seguridad de la Información
  - Trabajo publicado

TRABAJO FUTURO

- Disminuir la granuralidad de algunos controles para aumentar la precisión en la medición del riesgo residual
- Reestructurar las aplicaciones vulnerables para que la información de entrada se lea por un descriptor de archivo y no por la entrada estandar.
- Definir un modelo para los controles
- Para hacer multiplataforma el documento debe reestructurarse e incluirse el concepto de ABI
- Contemplar en los modelos variables propias de algunos compiladores (GOT, DTORS, ATEXIT) y de algunos lenguajes (apuntadores virtuales)

CONCLUSIONES

- Caracterización de 160 tipos de desbordamientos de buffer, 8 desbordamientos de enteros y 54 inyección de contenido sintáctico
- Identificación de 5 vulnerabilidades de validación de entrada en programas existentes (75.000 LOC)
- Actividades de divulgación a nivel regional, nacional e internacional con más de 150 asistentes
- Desarrollo y actualización de 16 metodos de explotación por medio de programación literal
- Identificación de falencias de 27 tipos de control actuales que permitió proponer 15 nuevas estrategias de control

CONCLUSIONES

- Construcción de 40 programas vulnerables y de explotación en lenguajes como Java, Python, C y Perl.
- Definición de las matrices de riesgo residual para los tres tipos existentes de vulnerabilidades de validación de entrada