Ofuscación focalizada para retrasar ataques 1-day

Francisco Blas Izquierdo Riera
Supervisor: Jonas Magazinius
Examinador: Andrei Sabelfeld

La situación actual

- La mayoría de gente usa el mismo SO
- Aparecen vulnerabilidades serias de tanto en tanto
- Los parches son pequeños por lo que el atacante puede centrarse en los cambios para ver que se ha arreglado

La fórmula

- Hacer más difícil encontrar el problema:
  - Marcar las funciones cambiadas
  - Añadir más funciones a las marcadas
  - Ofuscar sólo las funciones marcadas
  - Los cambios polimórficos generan cambios en el código generado
  - Los cambios de ofuscación hace más difícil comprender el código resultante

¿Cómo funciona la ofuscación?

- Coge un cacho de código
- Modifícalo para que sea más difícil de entender
- ???
- PROFIT!

¿Cómo funciona LLVM?

1) EL frontend convierte el código original en IR de LLVM
2) Se transforma el IR con pasadas de optimización
3) Se convierte el código a DAGs
4) El backend convierte los DAGs en instrucciones y las emite

¿Qué es eso del IR de LLVM?

- Una representación de los programas cercana al ensamblador.
- El código se divide en módulos
- Los módulos contienen funciones y globales
- Las funciones contienen bloques básicos
- Los bloques básicos contienen nodos phi seguidos de instrucciones y acabadas por un terminador
- Los nodos phi toman el valor según el bloque fuente.

¿Qué técnicas hay implementadas?

- Control Flattening de Wang (con adaptaciones)
- Ofuscación de Constantes
  - Extraer de array
  - Aritmética equivalente
- Reordenado
  - Reordenadp de instrucciones
  - Reordenado de bloques básicos
- Intercambiado de operadores

¿Cómo funciona la focalización?

- Se añade un parámetro a la función: la clave de ofuscación
- Se puede añadir a la declaración de la función en cualquier cabecera
- La clave tiene dos usos:
  - Provee el secreto para el CPRNG
  - Permite saber que funciones hay que transformar

¿Cómo va el control flattening?

- Se crea un bloque principal
- Se crean nodos phi para las referencias entre bloques
- Se mueven los nodos phi al bloque principal
- Se dividen los bloques con un terminador que no se puede procesar
- Se crea una phi en el bloque principal para elegir el destino según el origen
- Se añade un switch que depende del valor de la phi anterior para elegir el destino

¿Cómo va la ofuscación de constantes?

- Toma cualquier constante que se pueda ofuscar
- Elige una técnica:
  - Obtener de array: se obtiene el valor de un array
  - Aritmética: la constante se obtiene como el resultado de una operación
- Añade el nuevo código y reemplaza la constante por el resultado.
- Reofusca si lo deseas las nuevas constantes

¿Cómo se reordenan instrucciones?

- Reordena los nodos phi aleatoriamente
- Haz un mapa de dependencias de las instrucciones (incluyendo las dependencias ocultas por efectos laterales)
- Elige una de las instrucciones sin dependencias
- Emitela
- Elimina la instrucción del mapa
- Repite el proceso hasta que el mapa esté vacío

¿Cómo se reordenan bloques?

- Reordenar aleatoriamente manteniendo el bloque de entrada

¿Cómo se intercambian registros?

- Por cada operador binario conmutativo reordenar los operandos aleatoriamente

¿Qué significa “aleatorio”?

- Pseudoaleatorio para nosotros pero aparentemente aleatorio para el atacante.
- Usamos AES en modo CTR para generar datos “aleatorios”
- Se genera la clave para AES usando la clave de ofuscación, el nombre de la función el módulo y la pasada con AES en modo CMAC y una clave mágica: ABADCEBADABEBECEBADABEBECABECEA

¿Hay un pipeline recomendado?

- Sí, se recomienda el siguiente:
  - addmodulekey
  - propagatemodulekey
  - bbsplit
  - flattencontrol
  - obfuscateconstants
  - randins, randbb randfun, randglb and swapops

¿Cómo puedo usar esto?

- Crea una cabecera con las declaraciones de las funciones que vas a ofuscar con las claves de ofuscación correspondientes.
- Usa el parámetro -include para añadir la cabecera.
- Usa el pipeline clang | opt | clang para compilar.

¿Puede revertirse todo esto?

- El control flattening se puede revertir si se conocen las constantes
- La ofuscación de constantes puede revertirse mediante el cálculo de constantes
- Las transformaciones de reordenado pueden revertirse definiendo una ordenación
- Pero: ¡primero hay que volver del ensamblador al IR!
- No queremos evitar que se revierta el código eventualmente, sólo hacer más sencillo analizar las funciones originales

¿Pueden implementarse nuevas técnicas?

- Sí, el límite eres tú, el código es abierto.

¿Preguntas?