XII Jornadas STIC CCN-CERT

Ciberseguridad,
hacia una respuesta y disuasión efectiva

Capturando similitudes entre procesos de Windows usando *Bytewise Approximation Matching Algorithms*

CCN-CERT
centro criptológico nacional

Miguel Martín-Pérez

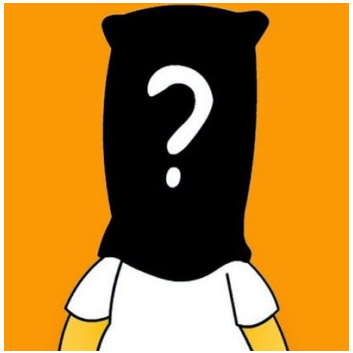
Universidad de Zaragoza

Email: mmarpe@unizar.es

Ricardo J. Rodríguez

Centro Universitario de la Defensa
Academia General Militar

Email: rjrodriguez@unizar.es



Mr. X

Centro Criptológico Nacional (CCN-CERT)

Email: ChunkyLover53@aol.com
(https://www.youtube.com/watch?v=xxuXt3XbYMk)

Agenda

1. Introducción
2. Algoritmos de *Bytewise Approximation Matching*
3. La herramienta ProcessFuzzyHash
4. Experimentos
5. Conclusiones

1. Introducción

- Respuesta a incidentes
  - ¿Cuáles son las razones detrás de un incidente de seguridad informática?
  - **Dar respuesta a las 6 W's** (*what, who, why, how, when,* y *where*)
  - 4 etapas (definidas por NIST):
    - **Preparación**
    - **Detección y análisis**
    - **Contención, erradicación y recuperación**
    - **Actividad post-incidente**
  - Análisis forense de ordenadores y redes es parte de *Detección y análisis*

1. Introducción

Análisis forense de ordenadores

- **Forense de discos**
  - **Analizar los dispositivos físicos del ordenador comprometido**
  - *Easy peasy*: apaga el sistema, coge el destornillador y quita los discos
  - Enchúfalos en un lector forense externo (**evita la escritura**), y usa herramientas forenses (e.g., OSS, FTK, EnCase, etc.)
- **Forense de memoria**
  - **Coger un volcado de memoria del sistema y analizarlo**
    - Contiene todos los elementos que se estaban ejecutando en el momento de la adquisición
  - *Memory artifacts*
    - Procesos en ejecución
    - Conexiones abiertas
    - Paquetes de red (totales/parciales)
    - …

1. Introducción

¿Cómo encontramos un malware?

- **Forense de discos**
  - **Recuperar todos los ficheros y calcular firma MD5/SHA1/SHA-256**
    - Conocidos como **hashes criptográficos** (formalmente, funciones *one-way*)
    - Proporcionan como salida un valor hash o *hex digest*
    - **Propiedad de "efecto avalancha"**
      - Una pequeña diferencia en el fichero (e.g., un bit cambiado) produce una salida totalmente diferente
      - Esto es, el mismo malware con un byte modificado tendrá un hash diferente y por tanto, **no será detectado**
    - **Similitud es {0, 1}**
      - Para los no matemáticos: los "{", "}" quieren decir intervalo entero

1. Introducción

¿Cómo encontramos un malware?

- **Forense de memoria**
  - *Easy peasy*: “Recuperar los ficheros y calcular firma MD5/SHA1/SHA-256”

INCORRECTO

[DEMO aquí]

<u>HECHO</u>: cryptohashes inválidos para calcular similitud entre procesos

- Fichero ejecutable (vista estática) vs. Proceso de ejecutable (vista dinámica)

1. Introducción

¡Espera! Pero… ¿por qué los hashes son diferentes?

- Formato de los ficheros binarios: *Portable Executable* (PE)
- **Direcciones de funciones importadas no están en el binario**
  - Sólo los nombres de las funciones
  - **Estos nombres se resuelven a direcciones específicas cuando el loader de Windows hace su trabajo**
  - Sección .idata o .import en el PE
- Los binarios ELF (de UNIX) se comportan de forma similar
  - A través de la sección .got

1. Introducción

¡Espera! Pero… ¿por qué los hashes son diferentes?

QUIZ RÁPIDO

Empezando con la "A", mecanismo de defensa software que ayuda a evitar ataques de control-flow hijacking

¡SÍ!

ASLR (*Address Space Layout Randomization*)

- **No sólo eso**: la *relocation* es también un problema
  - @ de símbolos pueden ser diferentes en ejecuciones posteriores

1. Introducción

Entonces, ¿cómo podemos calcular la similitud entre artefactos de memoria?

- *Approximate matching hashing*
  - Proporciona una **medida de similitud en el rango [0, 1]**
    - Para los no matemáticos: los "[", "]" significan intervalo real - es un porcentaje ☺
  - **Diferentes aproximaciones**
    - **Bytewise**: se basa únicamente en secuencia de bytes del artefacto digital
    - **Syntactic**: en estructuras internas
    - **Semantic**: en atributos contextuales del artefacto digital
  - **Dos interpretaciones:**
    - ***Resemblance***: la similitud mide cómo son de comunes
    - ***Containtment***: un objeto (más grande) contiene a otro

Aquí, nos centramos en algoritmos bytewise approximation matching

2. Algoritmos de Bytewise Approximation Matching

dcfldd

- Tipo *Block Based Hashing.* Propuesto por Nicholas Harbour (2002)
- Mejora del programa de copia dd, garantiza integridad de bloque
  - **Conocido como "dd on steroids"**
- Generación del hash
  - Divide la entrada en bloques
  - Concatena el crypto-hash de cada uno
- **Medida de similitud**: número de hashes en común / Total hashes
- **Problemas**:
  - **Insertar o eliminar un byte cambia todos los hashes sucesivos**
  - **Tamaño de la salida (tamaño de entrada y num. bloques usados)**

2. Algoritmos de Bytewise Approximation Matching

ssdeep

- Tipo *Context Triggered Piecewise Hash* (Jesse Kornblum, 2006)
- **Ampliamente utilizado** (e.g., análisis estático de malware)
- Combina dos hashes distintos, uno tradicional y otro rolling
  - **Tradicional**: 32 bit FNV-1a. Sólo usa los 6 bits menos significativos
    - Problema de mapear 32 bits en Base64
  - **Rolling Hash** (7 bytes): basado en Adler-32. Cuando se detecta el punto de disparo, se concatena salida del hash tradicional, tomando como entrada la ventana actual sobre la entrada (contexto)
  - **Objetivo**: generar de 32 a 64 contextos
- **Medida de similitud**: distancia de edición Damerau-Levenshtein

2. Algoritmos de Bytewise Approximation Matching

ssdeep

- **Tamaño de la salida:**
  - Dependiente de la entrada (de 67 a 140 caracteres)
- **Problemas**:
  - Límite de 64 contextos: último contexto contiene el resto
  - Sólo puede comparar sub-hashes del mismo tamaño de bloque
    - Genera hashes para 2 tamaños de bloque
    - **No se pueden comparar datos de tamaños muy diferentes**
  - **Colisiones**: contextos diferentes pueden tener mismo identificador

2. Algoritmos de Bytewise Approximation Matching

sdhash

- Tipo *Statistically-Improbable Features* (Vassil Roussev, 2010)
  - **Idea**: escoger *features* cuya ocurrencia es poco probable
- Generación de hash
  - Conjunto de Filtros Bloom con los crypto-hashes de *features* seleccionadas:
    1. Entropía *S* de subcadenas de 64 bytes, descarta $S < 100$ y $S > 990$
    2. Genera precedencia de cada *feature,* en función de *S*
    3. Cuenta las veces que una *feature* es la primera en tener la menor precedencia en ventanas de 64 valores
    4. Si el contador alcanza 16, genera el hash y lo introduce en el filtro Bloom
- **Medida de similitud**: porcentaje de *features* en común

2. Algoritmos de Bytewise Approximation Matching

sdhash

- Tamaño de la salida:
  - Longitud variable, entre un 2.6% y 3.3% del tamaño de la entrada
- **Problemas**:
  - Principio y final de datos no llegan a *features* seleccionables
  - La similitud es dada respecto al menor de los objetos:
    - O sea, misma similitud para comparar 2 objetos que para un fragmento y un objeto
  - En circunstancias límite, 2 ficheros con una diferencia de hasta el 20% pueden dar una similitud del 100%

2. Algoritmos de Bytewise Approximation Matching

TLSH

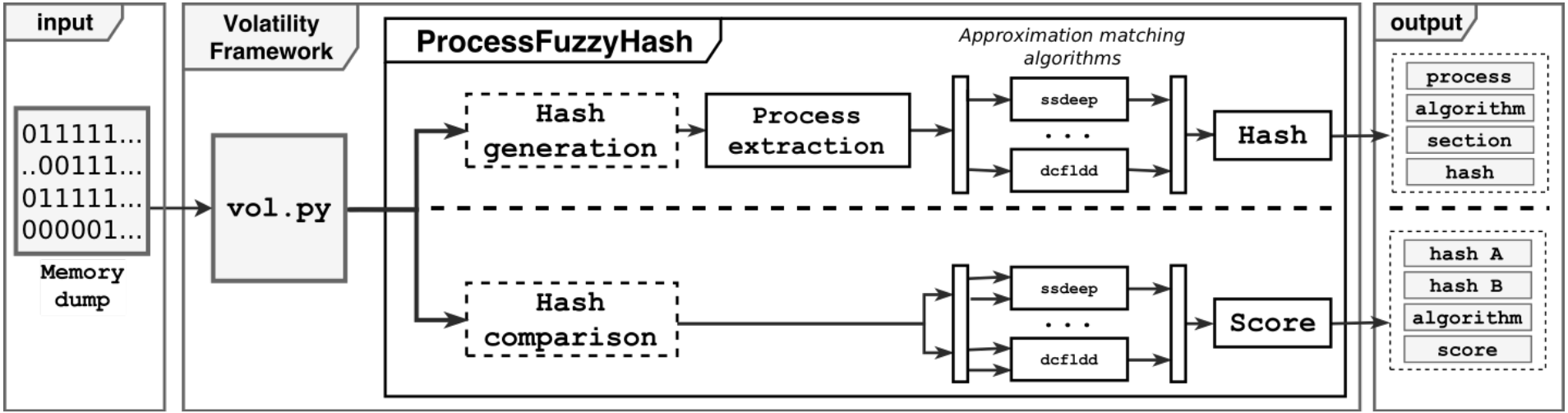
- Tipo *Local-Sensitive Hashing* (Jonathan Oliver *et al.*, 2013)
  - **Idea**: Mapear la entrada en “contenedores” por probabilidad y comparar el conjunto de “contenedores”
- Generación de hash
  - Dividido en dos partes
    - **Cabecera**: 3 bytes (checksum + log(tamaño) + ratio cuartiles)
    - **Cuerpo**: 32 bytes
      1. Genera identificadores (128) para todas las secuencias de 3 bytes
      2. Cuenta las apariciones de cada identificador
      3. Concatena el cuartil de los identificadores

2. Algoritmos de Bytewise Approximation Matching

TLSH

- **Medida de similitud**: distancia Hamming
  - Distancia entre cabeceras + distancia entre cuerpos
- **Problemas**:
  - La medida de similitud puede variar entre 0 y 1000 (o más)
    - **Normalización**: $S_{norm}(s) = \begin{cases} 100 - s, & s \leq 100 \\ 0, & s > 100 \end{cases}$
  - Resultados inversos (más próximo a 0 indica similitud)
  - Cubre toda la entrada, pero realiza un mapeado de rango pequeño
    - Susceptible de colisiones

3. La herramienta ProcessFuzzyHashing



- **Plugin de Volatility** (parte del repositorio oficial)
  https://github.com/volatilityfoundation/community/tree/master/ProcessFuzzyHash/ProcessFuzzyHash
- Dos actividades:
  - **Generación de hash**
  - **Comparación de hashes**

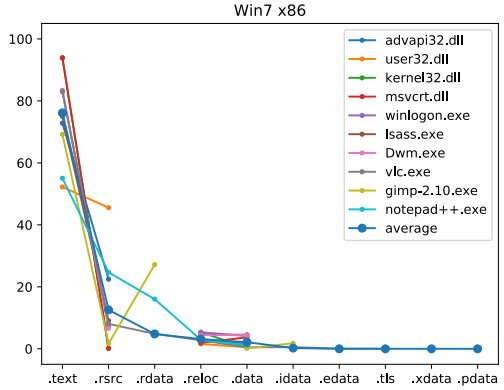
4. Experimentos

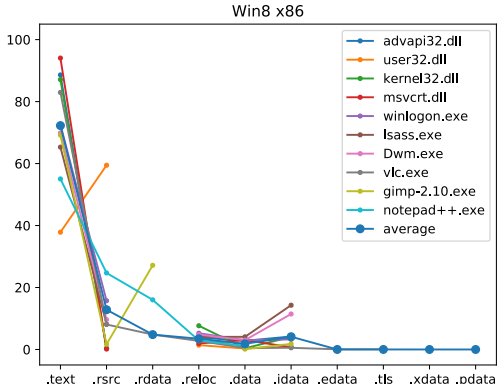
Entorno de experimentación

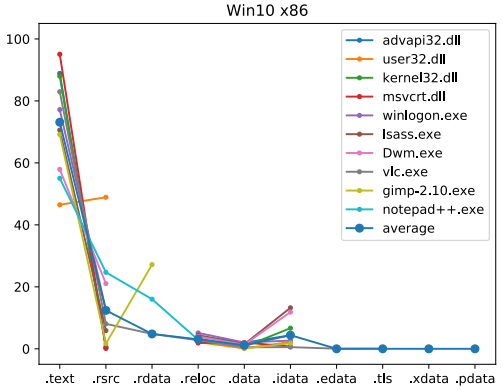
- Máquinas virtuales
  - 9 VMs, 10 volcados de memoria
- Diferentes sistemas operativos evaluados
  - Windows 7, Windows 8.1, y Windows 10
- Diferentes procesos:
  - **Aplicaciones de sistema**: winlogon, lsass, dwm
  - **Módulos de sistema**: kernel32, advapi32, user32, msvcrt
  - **Aplicaciones de usuario**: vlc, notepad++, gimp
- **Cálculo de límite**: maximizar precisión de cada algoritmo
- **Similitud: módulo y secciones PE específicas** (.text, .rsrc, .rdata)

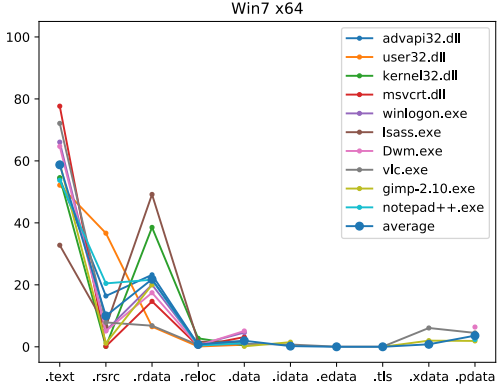
4. Experimentos

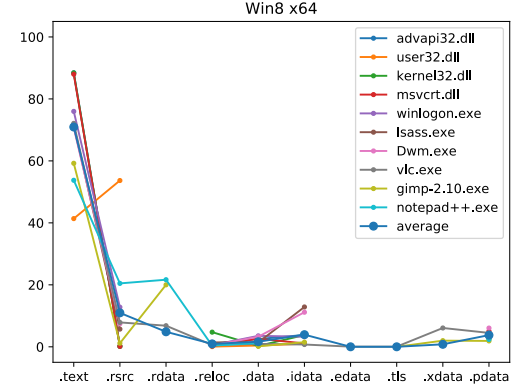
Tamaño de secciones

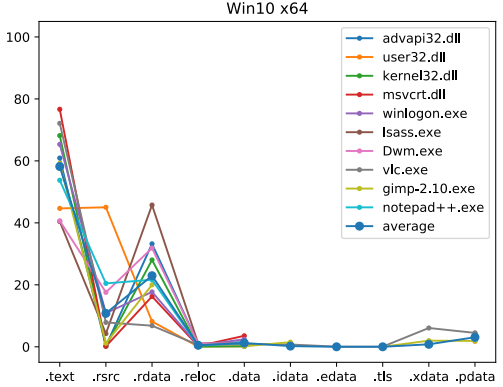




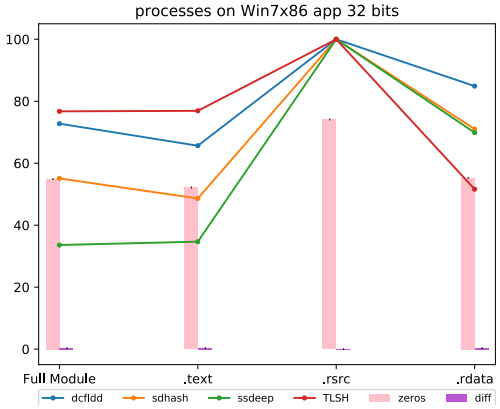


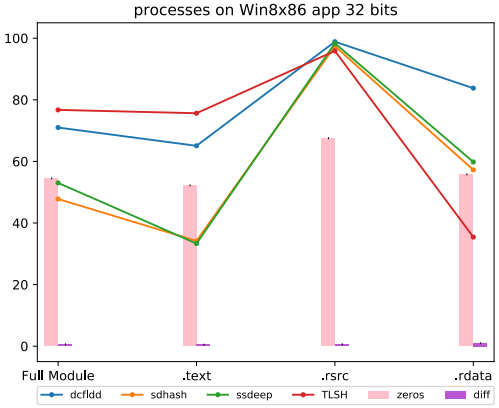


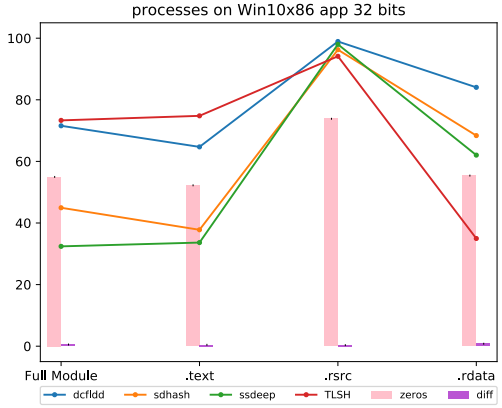


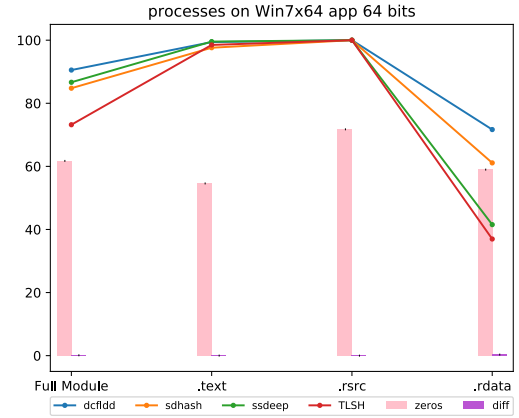


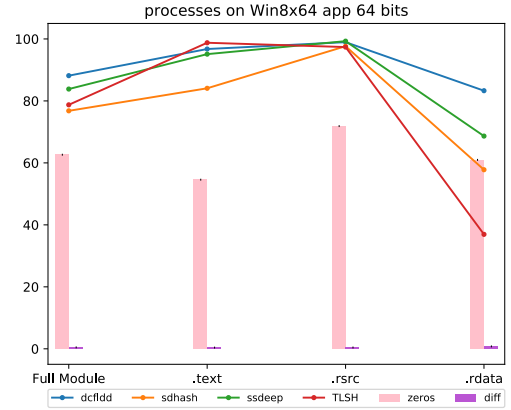
4. Experimentos

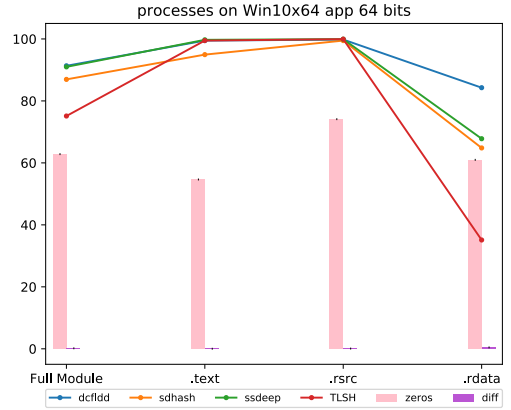




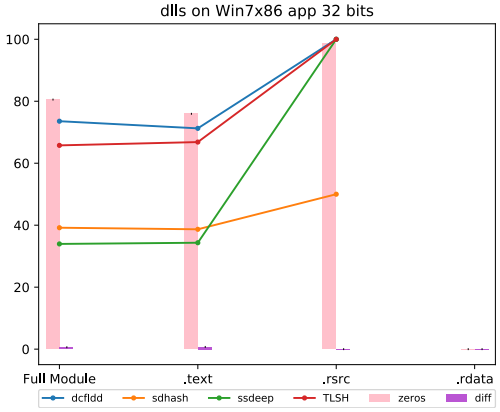


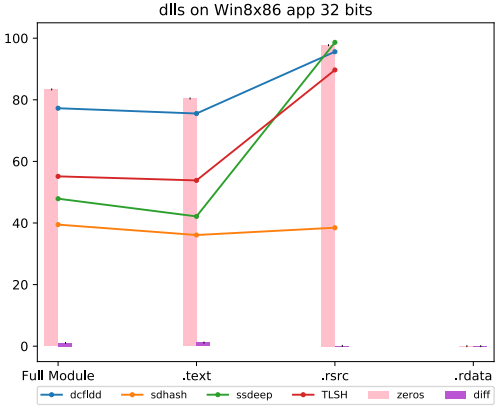


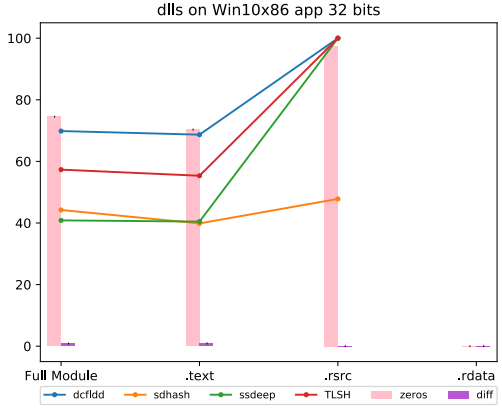


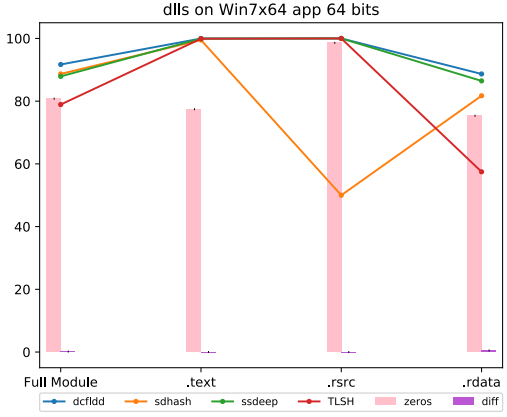


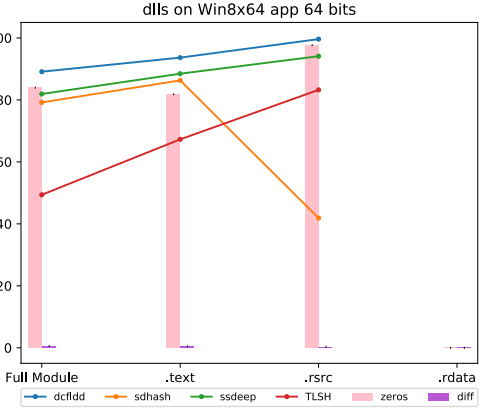
4. Experimentos

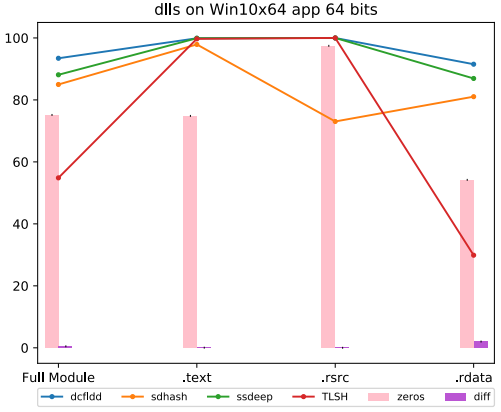




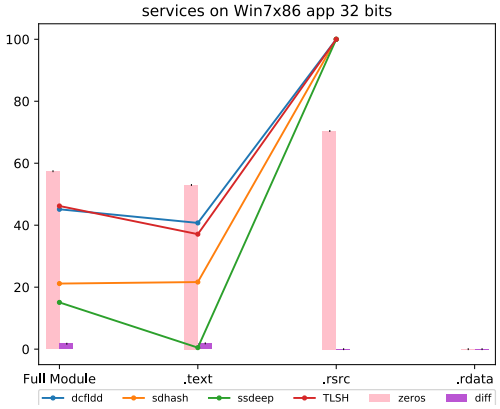


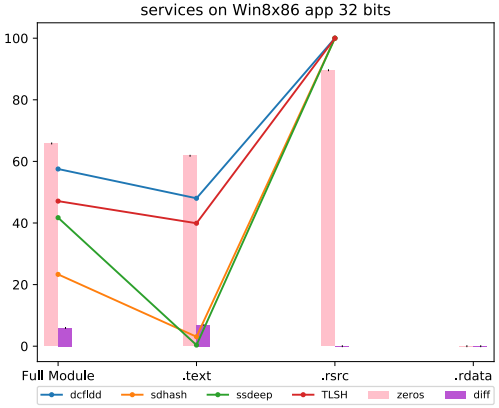


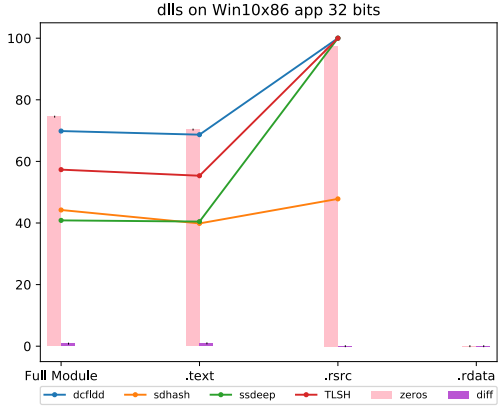


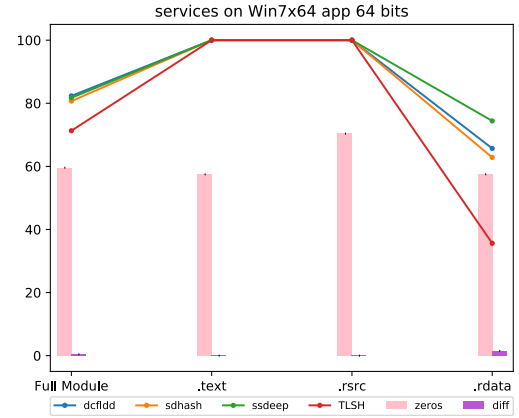


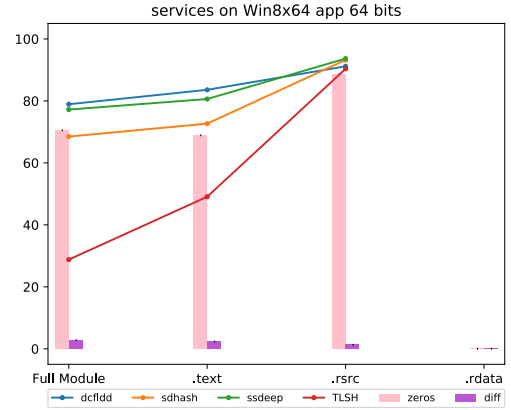
4. Experimentos

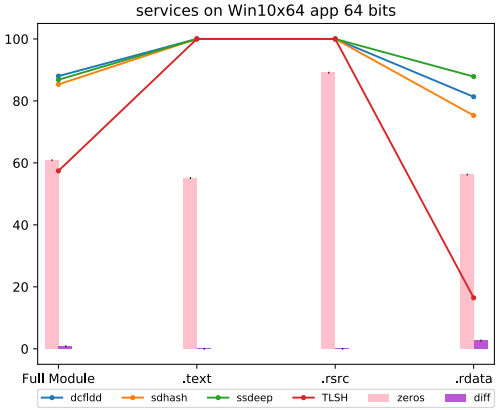












4. Experimentos

Resultados de la experimentación

(Precisión; Exhaustividad; Exactitud)

	Límite	Módulo	.text	.rsrc	.rdata
DCFLDD	>0	100%; 89%; 99%	100%; 89%; 99%	58%; 99%; 99%	100%; 90%; 98%
SSDEEP	>0	100%; 75%; 98%	100%; 68%; 98%	64%; 88%; 96%	100% 86%; 98%
	>47	100%; 70%; 98%	100%; 64%; 98%	100%; 88%; 99%	100% 80%; 97%
SDHASH	>0	100%; 99%; 99%	99%; 90%; 99%	100%; 56%; 97%	100%; 96%; 99%
	>5	100%; 96%; 99%	100%; 90%; 99%	100%; 56%; 97%	100%; 93%; 99%
TLSH	>100	68%; 83%; 97%	65%; 83%; 97%	48%; 86%; 93%	98%; 81%; 97%
	>40	100%; 50%; 97%	100%; 58%; 97%	100%; 81%; 98%	100%; 46%; 93%

- Precisión:
  - $P = \frac{TP}{TP+FP}$
- Exhaustividad:
  - $R = \frac{TP}{TP+FN}$
- Exactitud:
  - $A = \frac{TP+TN}{TP+FP+TN+FN}$

4. Experimentos

Conclusiones

- **Identificar un módulo por similitud a otros módulos del mismo binario es posible**
  - El Sistema Operativo no afecta a la similitud
  - La arquitectura sí afecta a la similitud
    - Arquitecturas de 64 bits obtienen similitudes mayores que de 32 bits
  - sdhash muestra los mejores resultados de identificación
    - Aunque los valores de similitud son más bajos

5. Conclusiones

- **Procesos contenidos en volcados de memoria**
  - Información muy dinámica, no se pueden aplicar técnicas tradicionales (i.e., hash criptográficos)
- **Algoritmos de *Approximate Matching Hashing* son útiles para calcular similitud entre artefactos digitales de memoria**
  - Cambios de SO no afecta a resultados, a diferencia de arquitectura (esperado…)
  - sdhash presenta mejores resultados
    - Valores de similitud bajos

¿Quizás algoritmos tipo bytewise no son los más apropiados?

XII Jornadas STIC CCN-CERT

Ciberseguridad, hacia una respuesta y disuasión efectiva

- **E-Mails**
  - info@ccn-cert.cni.es
  - ccn@cni.es
  - organismo.certificacion@cni.es
- **Websites**
  - www.ccn.cni.es
  - www.ccn-cert.cni.es
  - oc.ccn.cni.es
- **Síguenos en**

centro criptológico nacional