Introducción

Memoria Persistente:
- Mayor densidad
- No volátilidad
- Consumo de energía más bajo

Objeto de Memoria Persistente (PMO): - Una abstracción para guardar datos persistentes en memoria sin un acceso directo.
- Gestionado por el sistema operativo (namespace y permisos).

Motivación

- PMO programming model allows PMO sharing over time.
- This breaks inter-process isolation, making shared PMOs a new security vulnerability.
- Previous work focuses on making unauthorized accesses to PMOs difficult for the process accessing the PMO.

Problema de Investigación

- PMO corruption is persistent.
- Relaunching a process does not erase the effect of PMO corruption on its execution.
- PMO corruption by one process can affect any sharing process.
- Attacker can incrementally determine target data location for corruption across different runs.
- Attractive target for manipulation due to pointer-rich nature of PMOs.
- PMO accesses (via load/store) are not trapped by OS and hence not protected.

Contribution

- PMO programming model allows PMO sharing over time.
- This breaks inter-process isolation, making shared PMOs a new security vulnerability.
- Previous work focuses on making unauthorized accesses to PMOs difficult for the process accessing the PMO.

Fondo

- PMO Programming Interface:
  - attach() / detach() maps/unmaps a PMO to/from address space of a user process.
  - Once attached, PMO data is accessible via load/store instructions.
  - psync() persists a modified PMO in crash-consistent way.

PMO Sharing:

- PMO is outlived its creator process.
- A PMO can be shared by multiple processes over time.
- Simultaneous multiple readers.
- Write attach must be exclusive to other readers/writers.

Limitaciones de PMOs

- Two user processes, payload and victim, share a PMO over time.
- Victim has no memory safety vulnerabilities but the payload does.
- Adversary’s goal: Use payload to compromise victim process.
- Adversary’s knowledge:
  1. A PMO is shared.
  2. Data structure type.
  3. PMO layout.
  4. Trusted system software (i.e., OS).

Modelo de Ataque

- Control-data attacks: alter target program’s control data (e.g., return address and function pointer) to execute injected malicious code or stitched gadgets.

Clasificación de Puntadores

- Pointer classification:
  1. Absolute Pointer: Contains virtual address
     - Fast to dereference
     - Cannot be mangled
  2. Relative Pointer: Contains combination of PMO ID and offset
     - Translation table lookup is needed.
     - PMO relocation is less expensive.

Tipos de Ataque

- Control-data attacks: alter target program’s control data (e.g., return address and function pointer) to execute injected malicious code or stitched gadgets.

Defensas Posibles

- PMO Space Layout Randomization (PSLR): If enabled, randomizes the addresses (e.g., of fp and M in Fig. 2) making it hard for attacks to succeed.
- Data Execution Prevention (DEP) prevents code injection i.e. M in Fig. 2.
- Stronger defense is needed if PSLR or DEP are breached.

Detección y Frenado del Ataque PMO-based:

- Vulnerable process with legit PMO access can be used by an attacker to launch a cross-process attack on a victim via shared PMO. In doing so, a shared PMO becomes security vulnerability.

Referencias


Aknowledgement

This work is supported in part by ONR through award N00014-20-1-2750 and NSF through award CNS-1717425.

Table 1. Summary of PMO attacks

<table>
<thead>
<tr>
<th>Attack</th>
<th>Assumptions</th>
<th>Detection strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>No PSLR &amp; No DEP</td>
<td>Topology verification</td>
<td></td>
</tr>
<tr>
<td>PM2NVM pointers permitted</td>
<td>Topology verification</td>
<td></td>
</tr>
<tr>
<td>Denial of service</td>
<td>Hash re-computation and comparison</td>
<td></td>
</tr>
<tr>
<td>Corrupting decision-making data</td>
<td>IDE verification</td>
<td></td>
</tr>
</tbody>
</table>