Strengthening Self-Checksumming via Self-Modifying Code

Jonathon T. Giffin, Mihai Christodorescu, Louis Kruger

Computer Sciences Department
University of Wisconsin

{giffin,mihai,lpkruger}@cs.wisc.edu
User name:

Initials:

Organization:

In the boxes below, type your 25-character Product Key. You'll find this number on the yellow sticker on the back of the CD case.

Product Key: _______ - _______ - _______ - _______ - _______
Self-Checksumming

• Program contains code to checksum parts of its own code.
Self-Checksumming

- **Integrity Verification Kernels** [Aucsmith 1996]
  - Multithreaded, self-checking, checksumming components

- **Testers and correctors** [Horne et al. 2001]

- **Network of guards** [Chang & Atallah 2001]
  - Many overlapping checksumming components
Assumptions

The attacker cannot identify all relevant checksum code within the protected program.

The attacker runs the protected program at full speed or with only a reasonable slowdown.

Self-checksumming programs execute on a commodity von Neumann machine.
Memory Architectures

von Neumann architecture

- Unified code & data memory
- Memory addresses unique

[von Neumann 1945]
Memory Architectures

Harvard architecture

- Separate code & data memory
- Memory addresses duplicated

[Aiken & Hopper 1946]
Page-Replication Attack

[Wurster et al. 2005]
Page-Replication Attack

[Wurster et al. 2005]
Page-Replication Attack

CPU

Write
Read

Fetch

D-RAM
Data
Code
Genuine

I-RAM
Code
Altered

[Wurster et al. 2005]
Attack Detection

Observation:

Writes to code affect program differently depending upon memory architecture

Use self-modifying code to detect page-replication attack
Self-Modifying Code

CPU

RAM

Data

I_1

Write
Read
Fetch
Self-Modifying Code

Both read & fetch use the rewritten instruction $I_2$
Attack Detection

![Diagram showing CPU interactions with I-RAM and D-RAM, highlighting the process of attack detection.](image)

- **CPU** interacts with **I-RAM** and **D-RAM**
- **Fetch** (from I-RAM)
- **Read** (from D-RAM)
- **Write** (to D-RAM)

**I-RAM**:
- **Genuine**
- **Altered**

**D-RAM**:
- **Data**
- **I₁**
Read / fetch mismatch detects page-replication attack

Attack Detection

CPU

D-RAM

Data

I$_2$

Genuine

I-RAM

I$_1$

Altered

Write

Read

Fetch
Resistance to Attack

Attacker splits writes to code to update both D-RAM & I-RAM

CPU

D-RAM
Data
I_2
Genuine

Write

I-RAM
I_2
Altered
Resistance to Attack

• Split writes requires attacker to emulate writes to code
  – Efficiently done via memory page protection bits
Resistance to Attack

- Split writes requires attacker to emulate writes to code
  - Efficiently done via memory page protection bits

- Make code writes appear identical to data writes
  - Interleave code and data
  - Successful attack requires emulation of all writes
Drawbacks

• Increases debugging complexity
  – Add self-modifying code in final development stage

• Requires writable code pages
  – Use alternative attack detection/prevention techniques

• Harvard caches must be kept consistent
Cache Coherency

CPU

Write

Read

Fetch

RAM

Data

Code
Conclusions

Self-modifying code ...

... detects page-replication attacks ...

... efficiently ...

... in a way robust up to emulation attacks ...

... restoring the previous viability of self-checksumming.
Questions?

Contact the authors:

Jonathon T. Giffin giffin@cs.wisc.edu
Mihai Christodorescu mihai@cs.wisc.edu
Louis Kruger lpkruger@cs.wisc.edu

Computer Sciences Department
University of Wisconsin