# A Framework for Understanding Dynamic Anti-Analysis Defenses

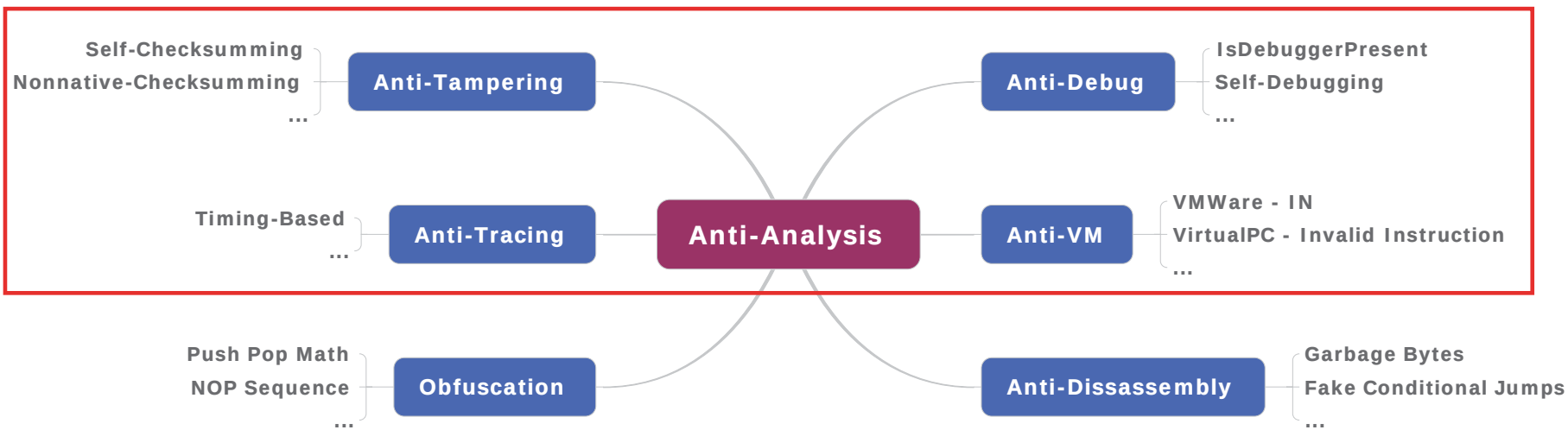
Jing Qiu  Babak Yadegari  **Brian Johannesmeyer**
Saumya Debray  Xiaohong Su

Harbin Institute of Technology, Harbin, China
The University of Arizona, Tucson, US

Dec. 9, 2014

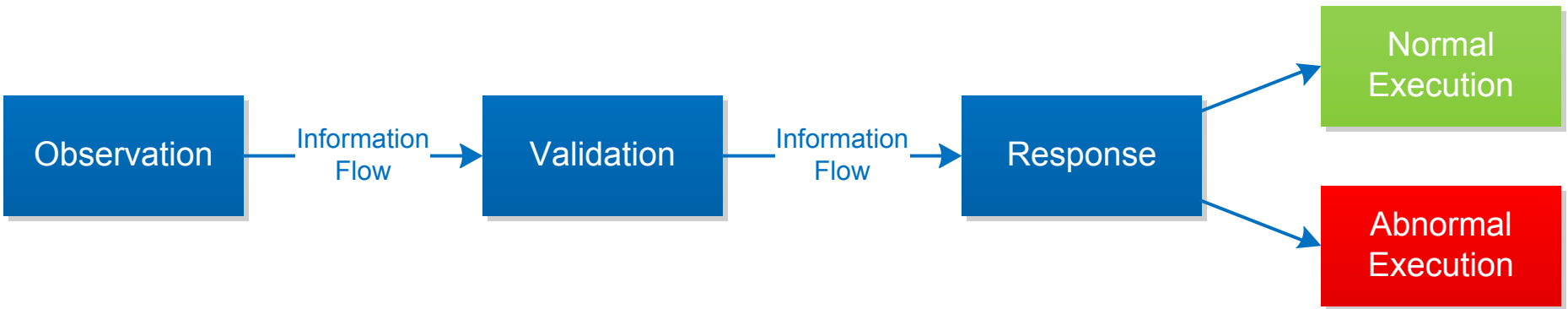
# Motivation

Malware combine a variety of defenses to avoid detection and hinder analysis.

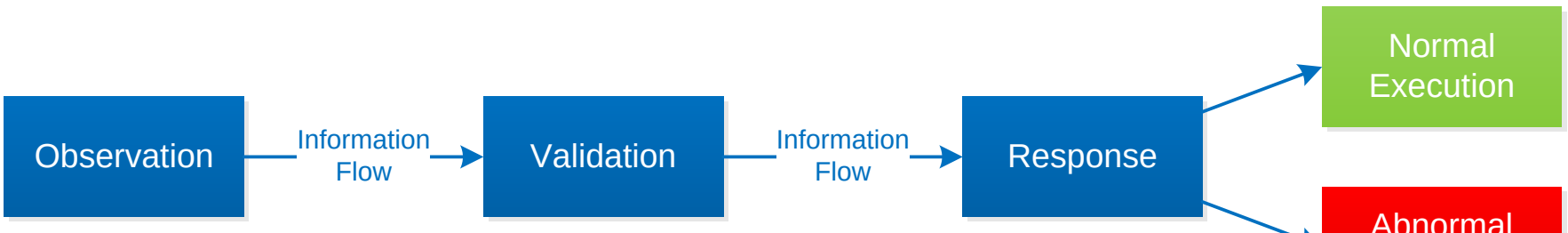


# Framework

We propose a general information-flow-based framework to detect anti-analysis defenses.



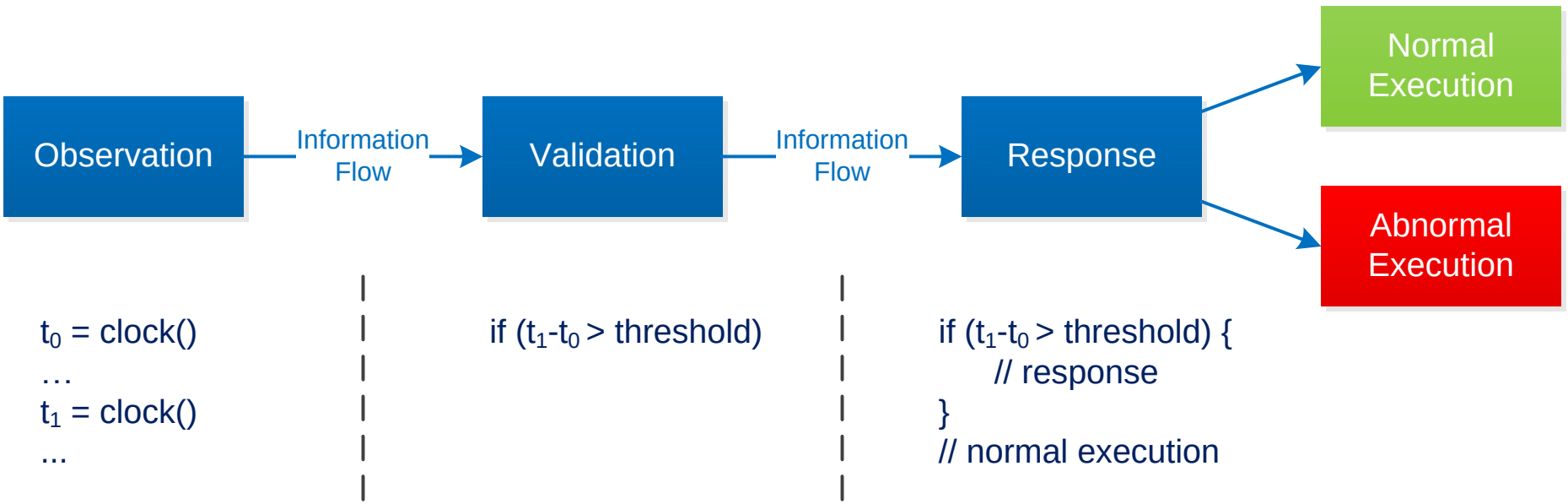
# Self-Checksumming



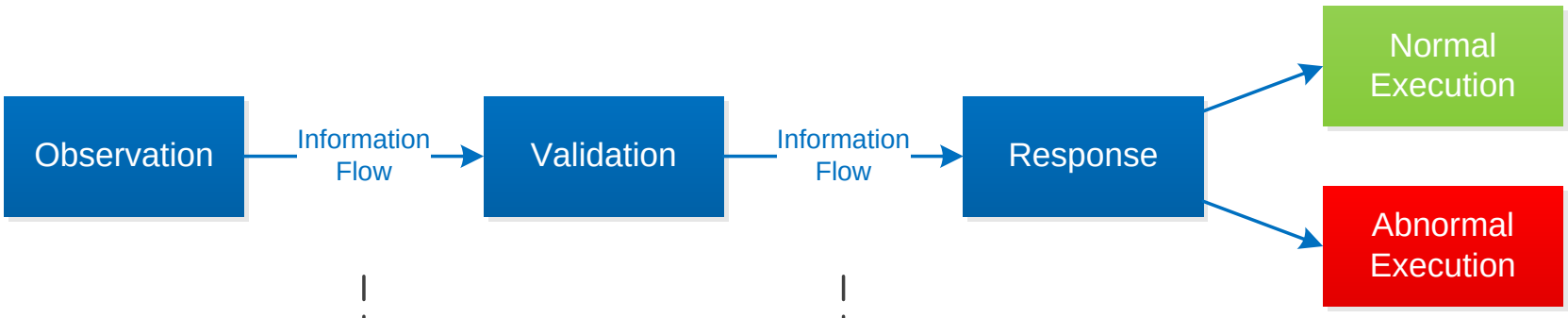

Computing a checksum

- Simple comparison
- Checksum-based code unpacking

# Timing-Based Anti-Emulation



# Anti-Debugging using ProcessDebugPort

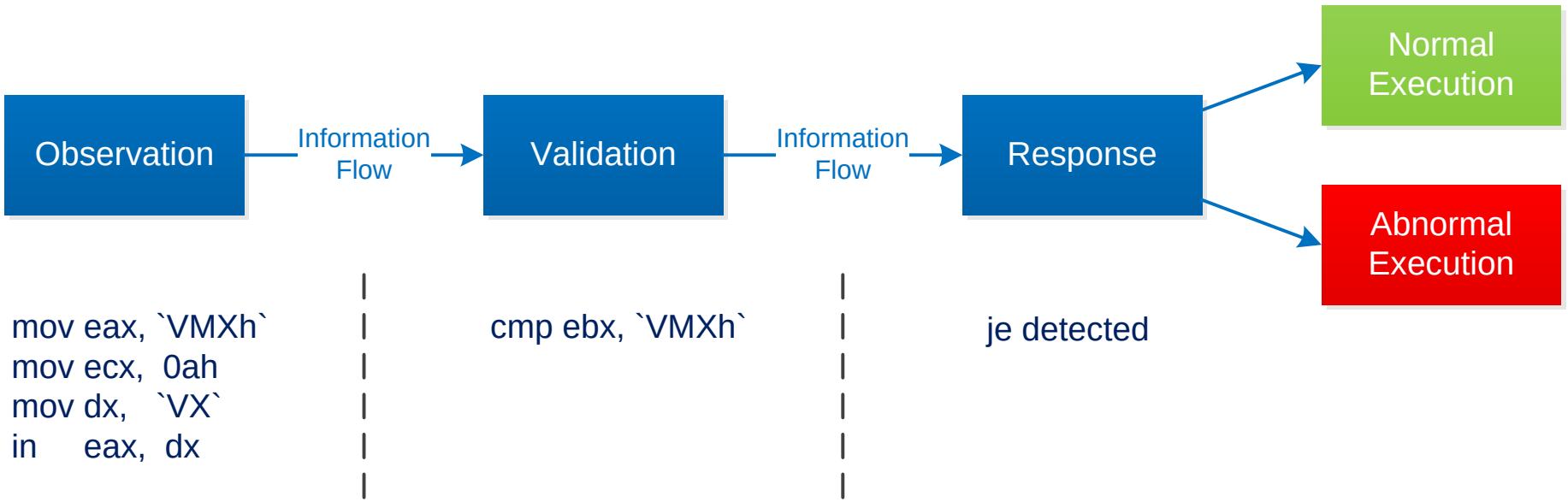



```
NtQueryInformationProcess(-1,
ProcessDebugPort, &retVal, ...);
```

```
if (retVal != 0)
```

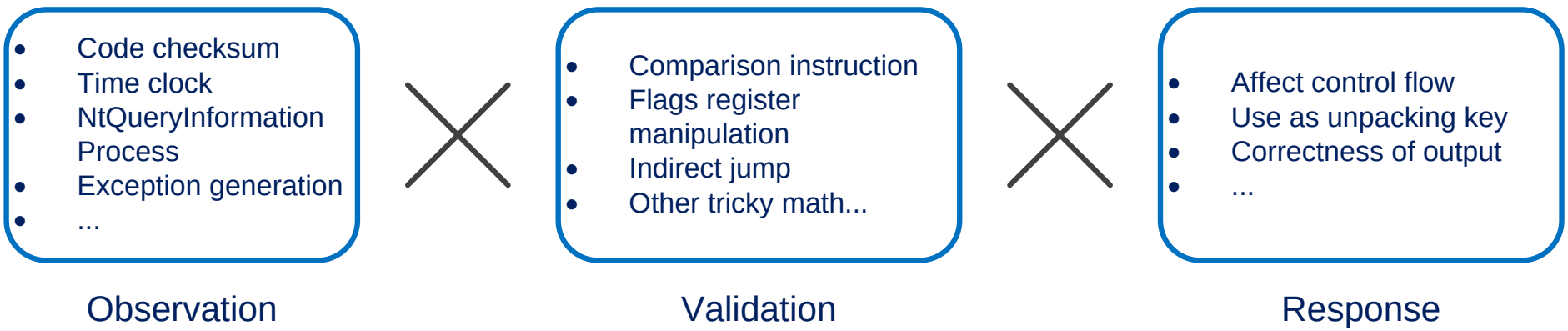
```
if (retVal !=0 ) {
    // Debugger Detected
} else {
  // Normal execution
}
```

# Anti-VM - Detect VMWare

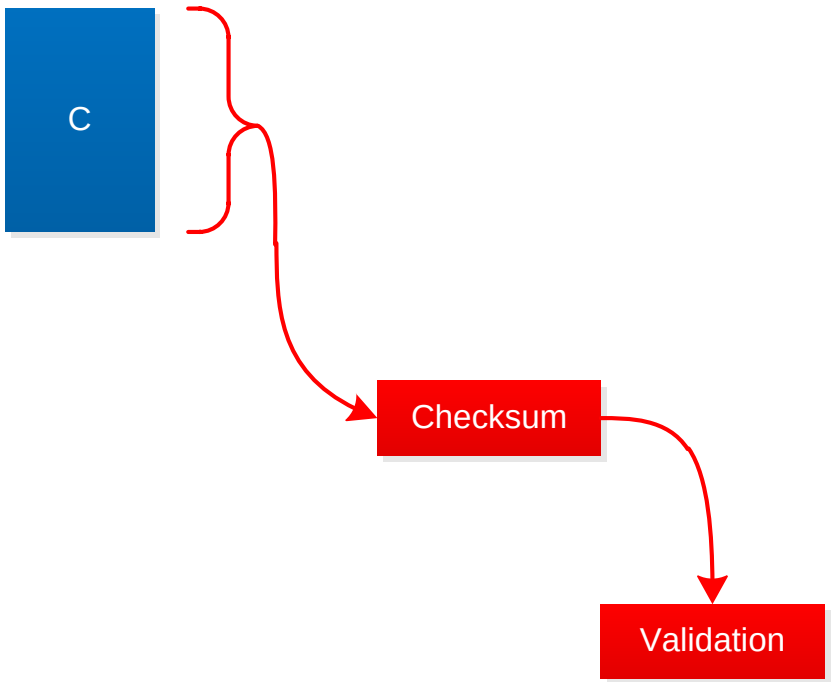


# Design Space

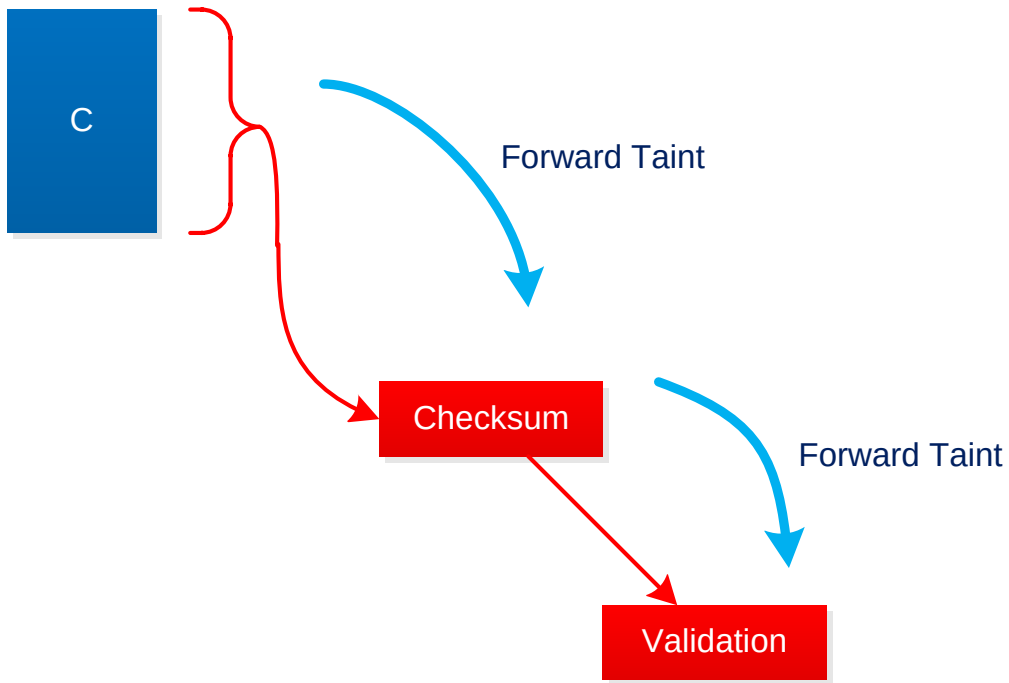
Stages of the framework can be used in different combinations to create and understand new defenses.



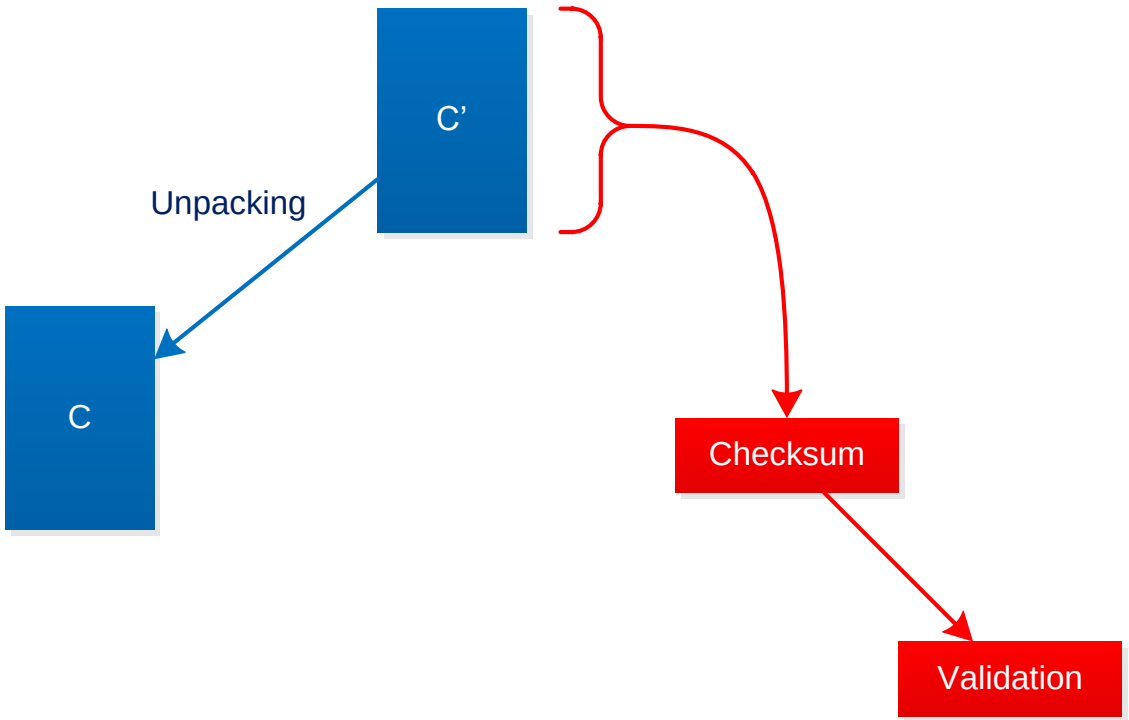
# Detection: Self-Checksumming (1/6)



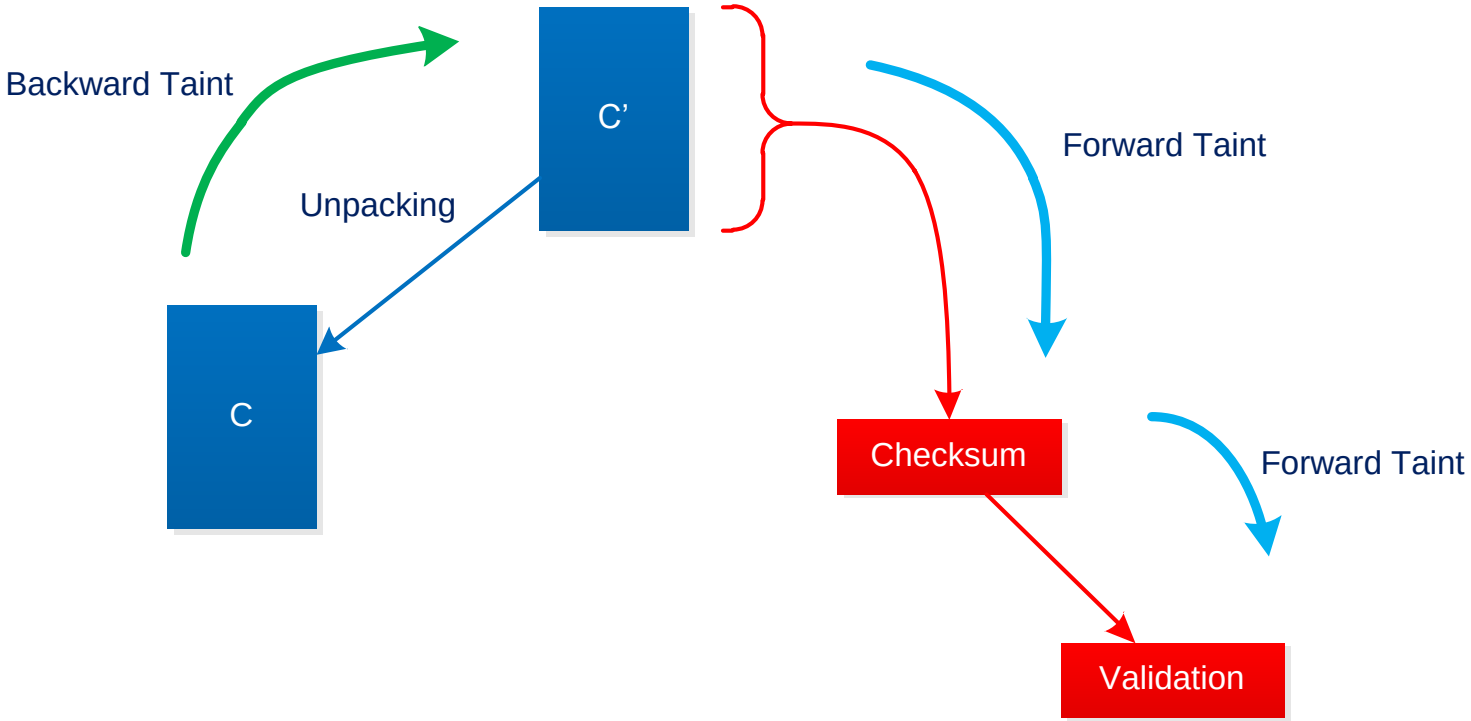
# Detection: Self-Checksumming (2/6)



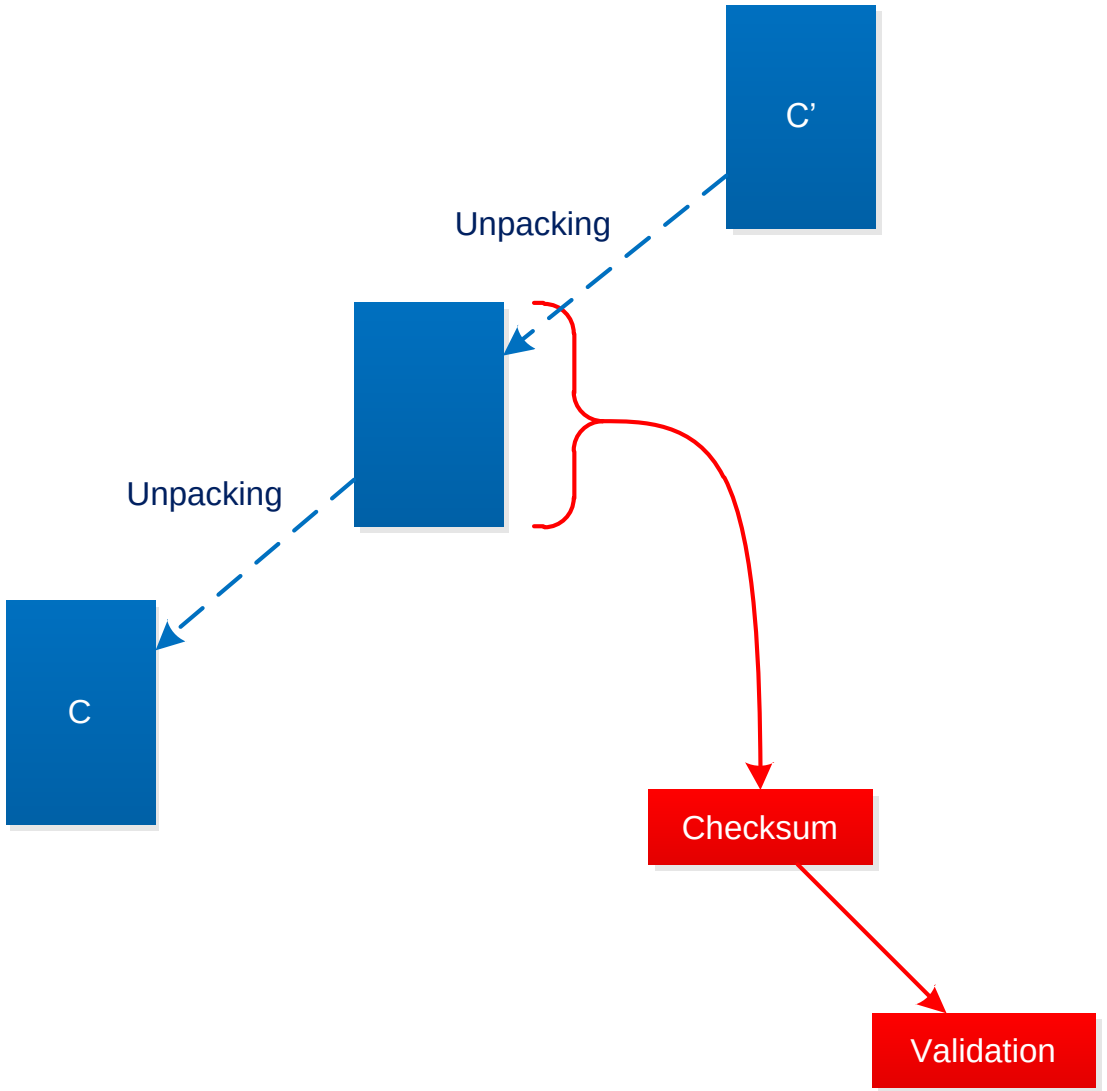
# Detection: Self-Checksumming (3/6)



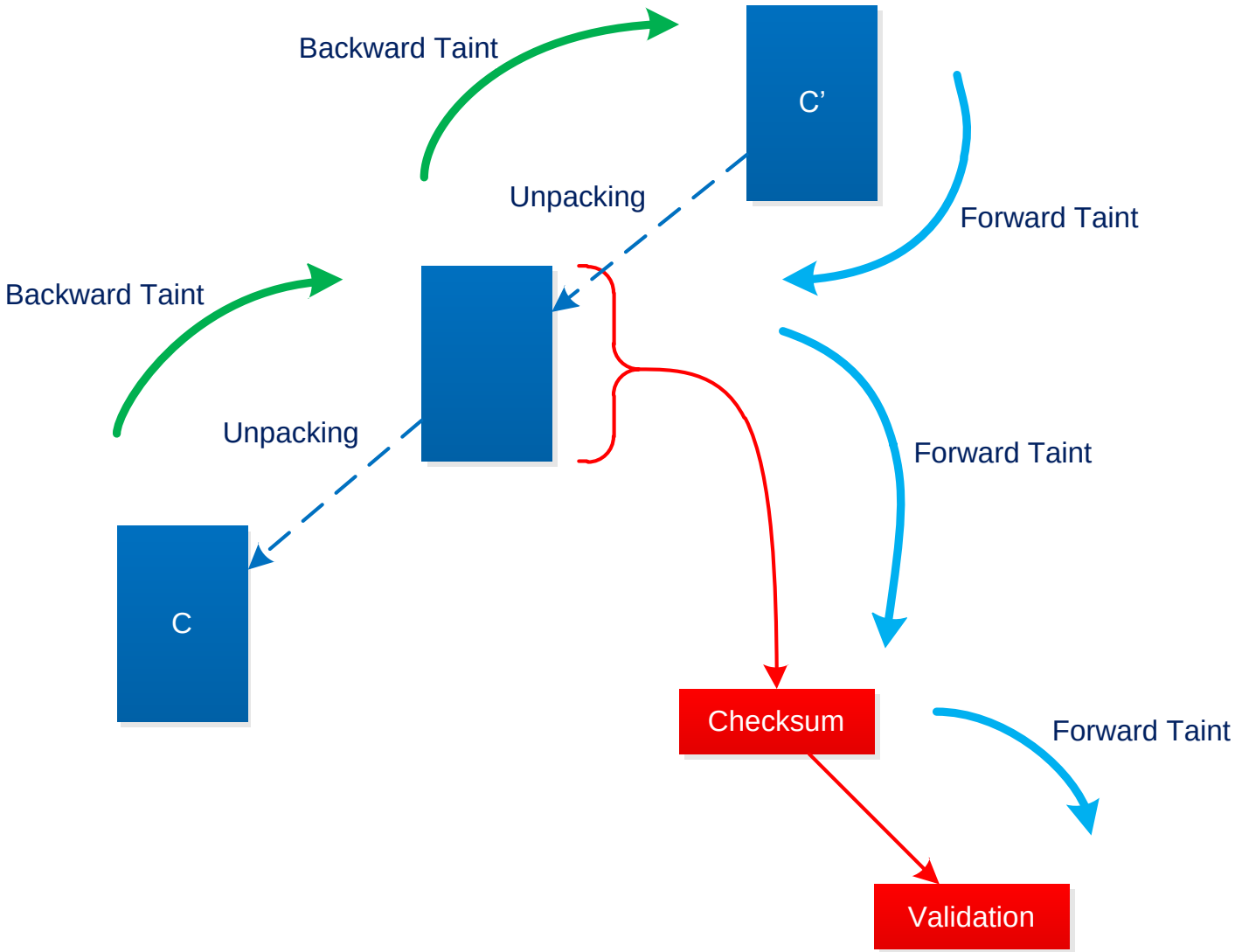
# Detection: Self-Checksumming (4/6)



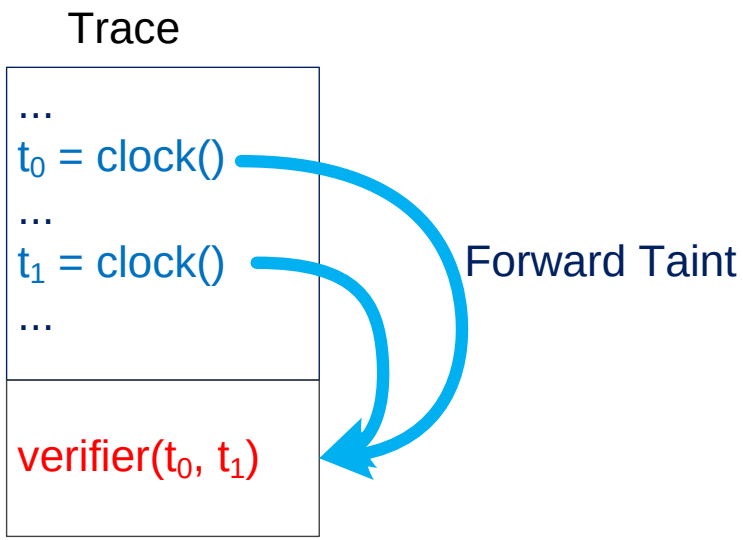
# Detection: Self-Checksumming (5/6)



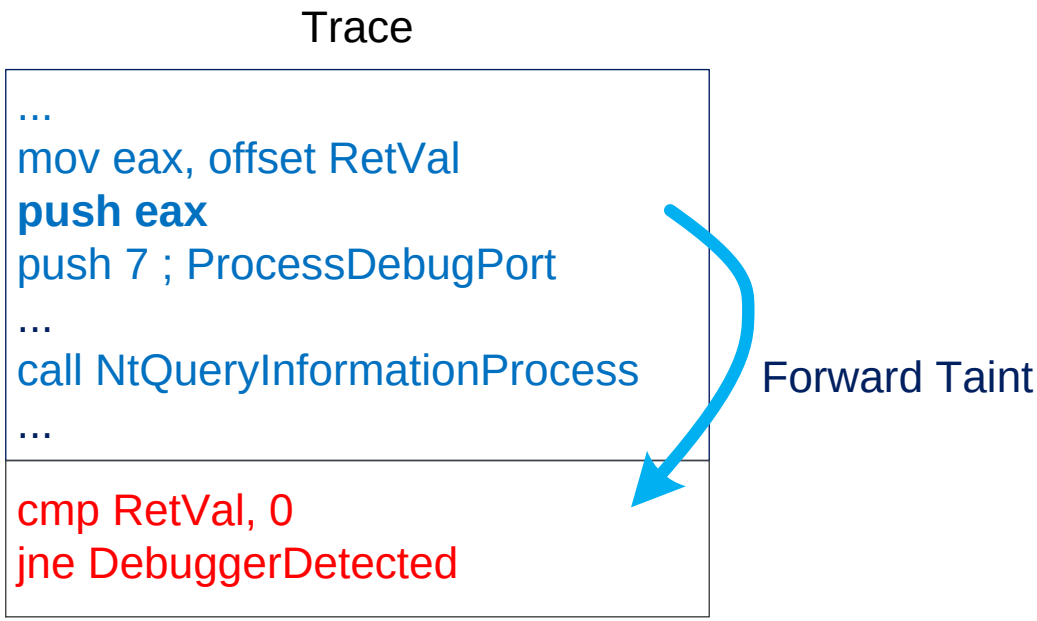
# Detection: Self-Checksumming (6/6)



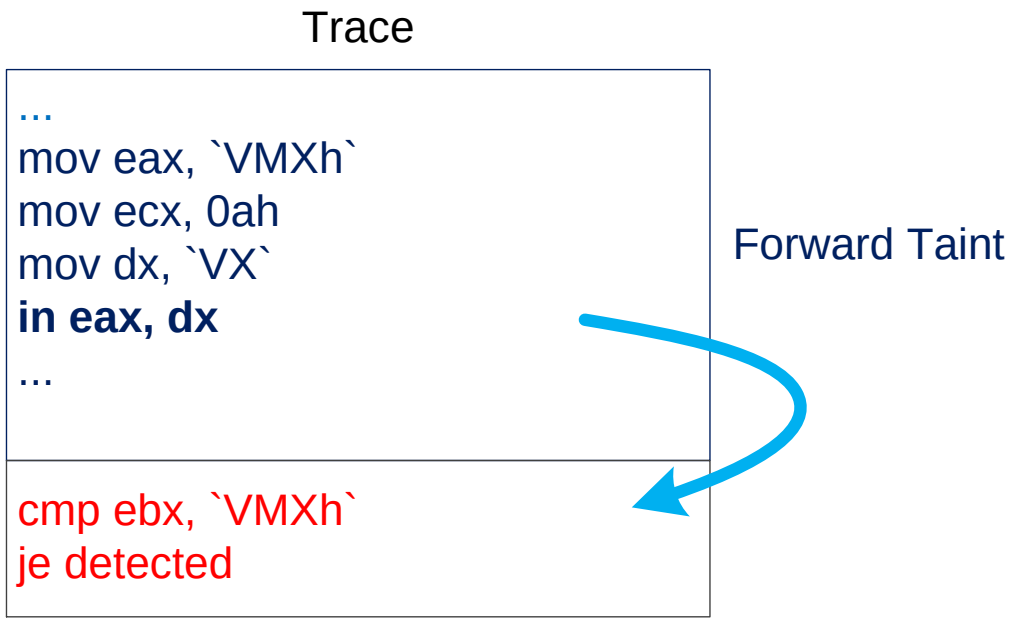
# Detection: Timing-Based Anti-Emulation



# Detection: Anti-Debugging using ProcessDebugPort



# Detection: Anti-VM - VMware



# Evaluation

## Setup - Samples

- **50-guards**: 50 different interleaving self-checksumming guards
- **decrypt-key**: a checksum as a code decryption key
- **chksum-md5**: a checksum to generate an MD5 initialization constant
- **time-md5**: time-based defense
- **timing-themida** & **timing-obsidium** packed with Themida and Obsidium

# Result

Table 1: Evaluation Result

| Program | No. of Guards | | Analysis Time (sec) | Instructions in Trace |
|---|---|---|---|---|
| | Found | Ground Truth | | |
| 50-guards | 50 | 50 | 95 | 2,702,679 |
| decrypt-key | 1 | 1 | 8 | 124,472 |
| chksum-md5 | 1 | 1 | 4 | 296,138 |
| time-md5 | 4 | 4 | 5 | 226,855 |
| timing-obsidium | 0 | Unknown | 347 | 27,461,928 |
| timing-themida | 2 | Unknown | 223 | 9,304,222 |

# Limitation

- Low code coverage
  - The code coverage could be improved by multiple path exploration
  - or generating inputs by symbolic execution
- Large scale trace files
  - Recording relevant instructions
  - or parallel processing

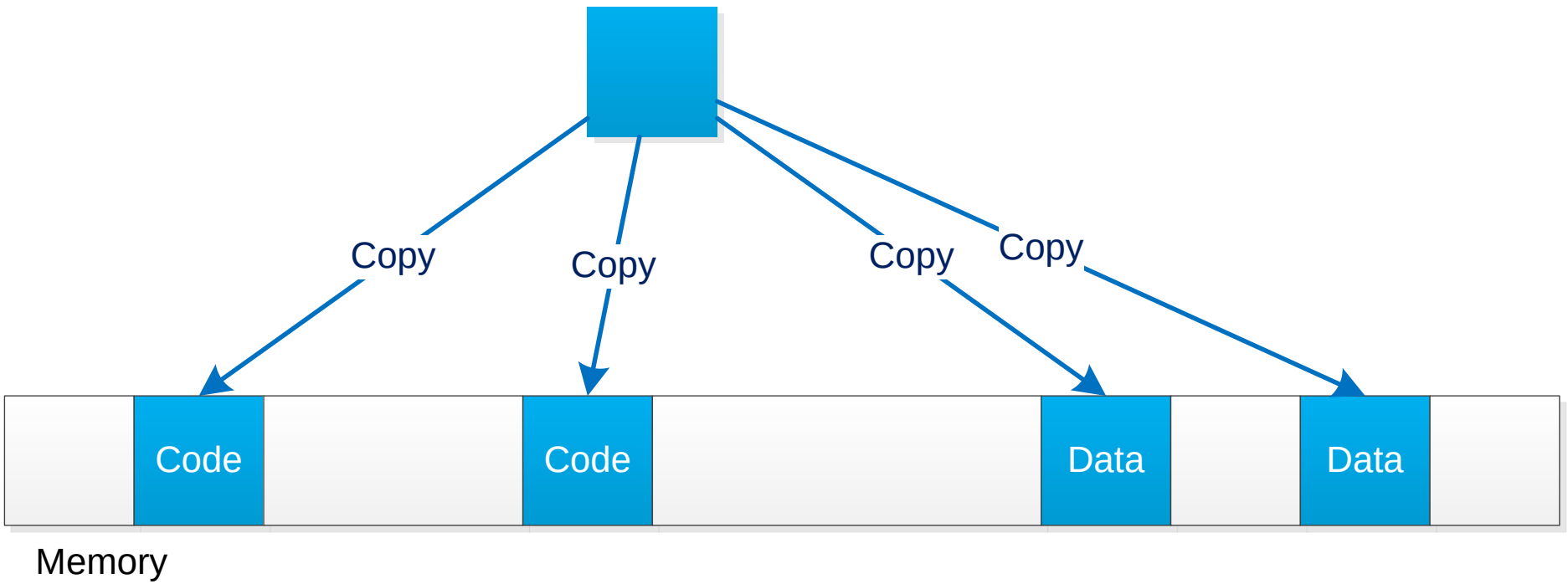
# Challenges - Implicit Flow

## Example

```
x=0; z=0;
if(y=0) then x=1; else z=1; endif
if(x=0) then w=0; endif
if(z=0) then w=1; endif
```

# Challenges - False Positives

Self-Checksumming: Data compression



# Challenges - False Positives

Timing-based anti-emulation: Benchmark programs

## Example

```
t = clock()
...
t = clock() - t
if (t < 10) score = 100;
else if (t < 20) score = 90;
...
```

# Conclusion

- We describe an information-flow-based framework for understanding a wide variety of anti-analysis defenses
- Self-checksumming, timing-based anti-emulation, anti-debugging and anti-VM defenses are instances of this framework.
- Experimental results shows that this framework is effective.

# Related Work

- In 2005, Wurster *et al.* employs hardware assisted techniques to bypass the self-checksumming defense.
- In 2005, Giffin *et al.* show that self-modifying code can be used to detect the attack Wurster *et al.* proposed.
- In 2008, Brumley *et al.* use a combination of dynamic binary instrumentation and mixed symbolic and concrete execution, to identify behavior that is dependent on environmental triggers.
- In 2006, Crandall *et al.* use a combination of VM-based timer perturbation and symbolic execution to discover time bombs in malware.
- In 2010, Lindorfer *et al.* and Balzarotti *et al.* discuss detecting environment-dependent behavior in native malware by comparing multiple executions in different environments.

# End

Questions?

# Code: decrypt-key

```
checksum = compute_checksum(CODE);
for(i = 0; i < size; i++)
    CODE[i] -= checksum;
```

# Code: chksum-md5

```
int cksum = compute_checksum(CODE);
...
//This value should be 0x67452301;
mdContext->buf[0] = chksum + 0x6740E9CB;
...
printf("%s", md5_str);
...
```

# Code: timing-md5

```
DWORD t1 = GetTickCount();
MD5();

if (GetTickCount() - t1 > 50)
{
        printf("I am traced.\n");
}
```