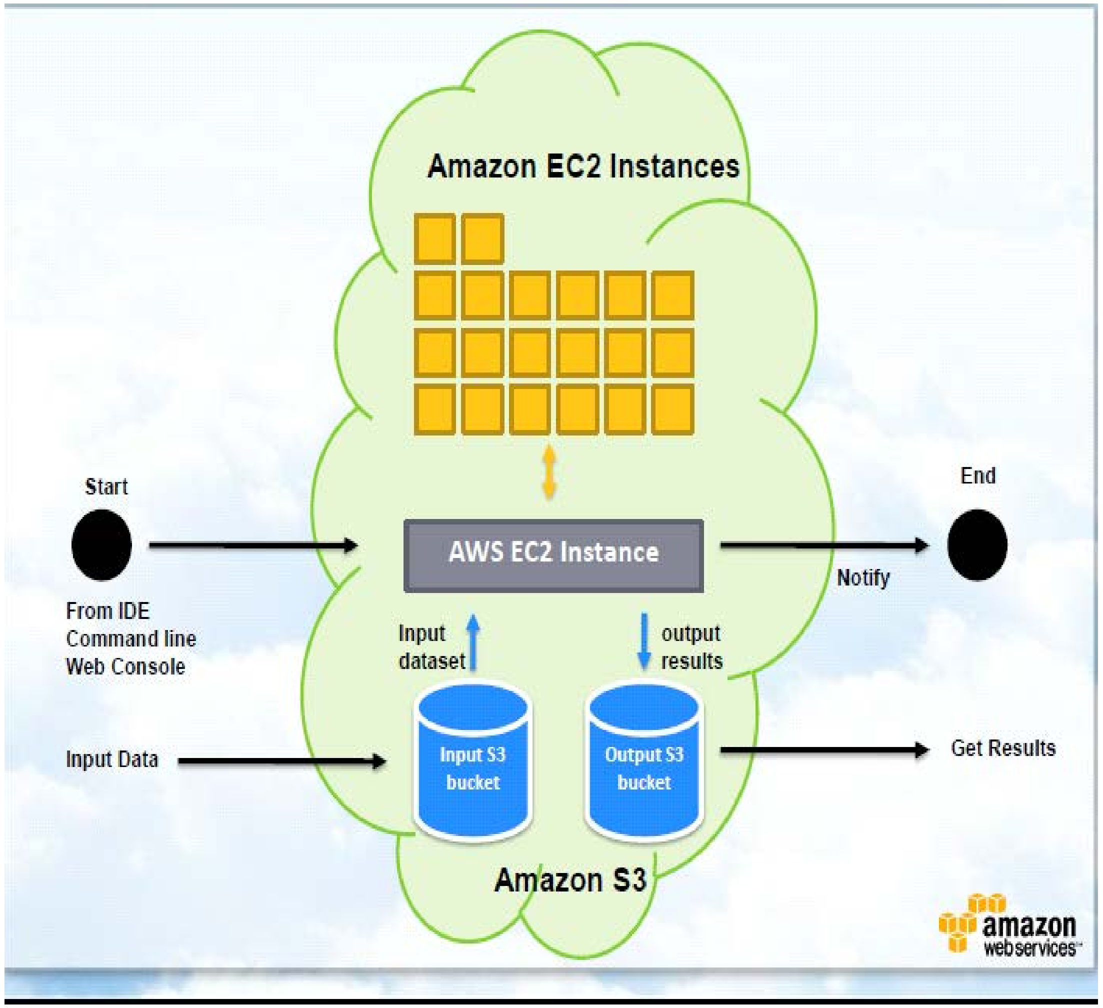
# ZeroVM Backgroud

Prosunjit Biswas
Institute for Cyber Security
University of Texas at San Antonio

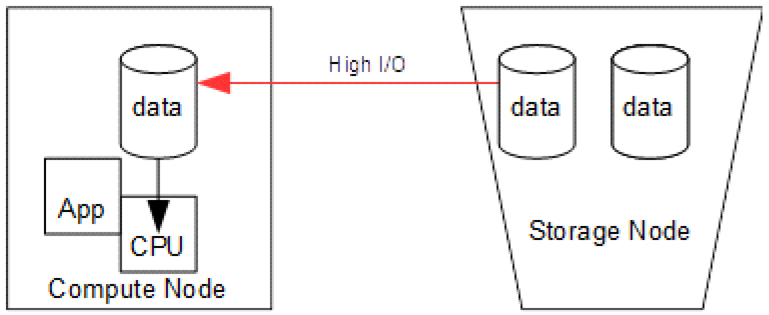
April 23, 2014
Institute of Cyber Security, ICS @ UTSA

## Motivation Behind ZeroVM

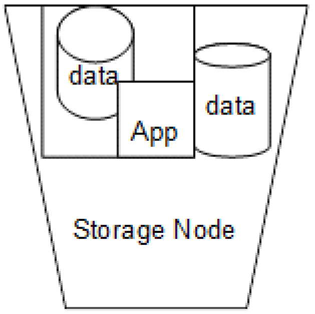


1. In Amazon map/reduces a considerable amount of overhead was due to fetching the data from s3 to EC2 Instances and put it back to s3.
2. The overhead was hurting when the customers need to remake to cluster and do the map/reduce again.
3. A significant amount of customer's money was spent due to moving the data back and forth.

## Motivation Behind ZeroVM(continued)



Challenge with High I/O



Challenge with Application Isolation

1. can we bring to Application to the data(very limited I/O overhead)?
2. How can we ensure no harm even if the application is malicious?

## What is ZeroVM

*ZeroVM is an* ***open–source lightweight virtualization platform*** *based on the Chromium Native Client project.*

## ZeroVM Properties

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - ➢ Channel based I/O
   - ➢ Predefine socket port / network
   - ➢ Restricted Memory Access
   - ➢ Limited Read/ Write (in bytes)
   - ➢ Limited life time / Predefined timeout

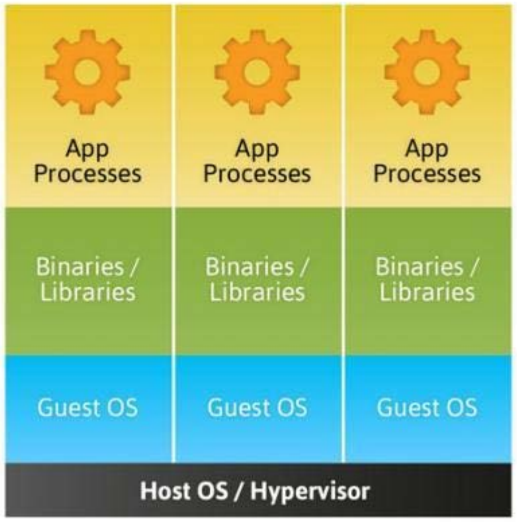
## ZeroVM Properties

1. ZeroVM virtualizes Application not Operating System.

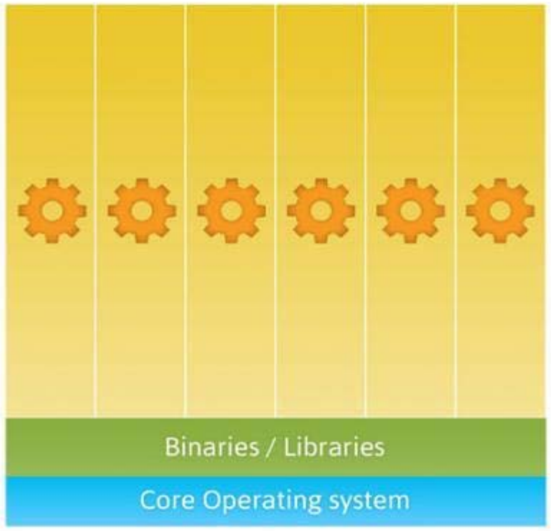
2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - Channel based I/O
   - Predefine socket port / network
   - Restricted Memory Access
   - Limited Read/ Write (in bytes)
   - Limited life time / Predefined timeout

## Popular Virtualizations



OS Level Virtualization



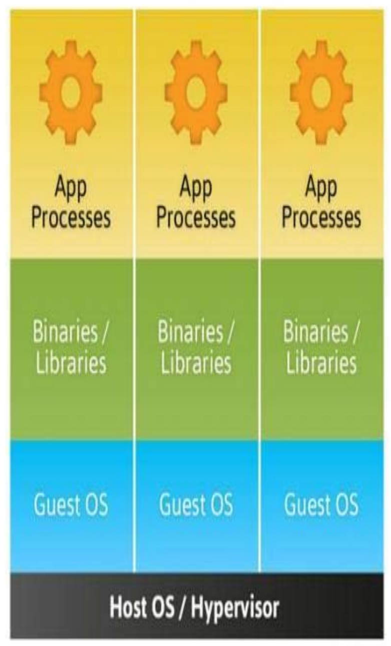
Process Level Virtualization

**1.** ZeroVM virtualizes Application not Operating System.

2. Does zeroVM uses process level virtualization ?

No

## Popular Virtualizations



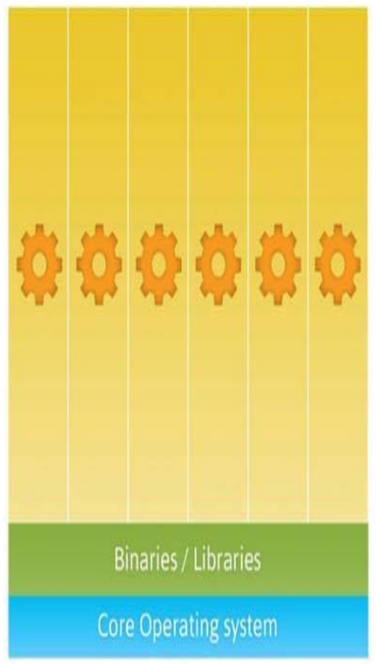
**Pros:**

1. Complete Isolation
   - Dedicated V. Memory
   - Dedicated V. Storage
   - Dedicated V. CPU
2. Flexible Architecture
   Almost all OS is supported
3. Fault Tolerance

**Cons:**

1. High Resource Overhead
2. High Maintenance Cost.

OS Level Virtualization



**Pros:**

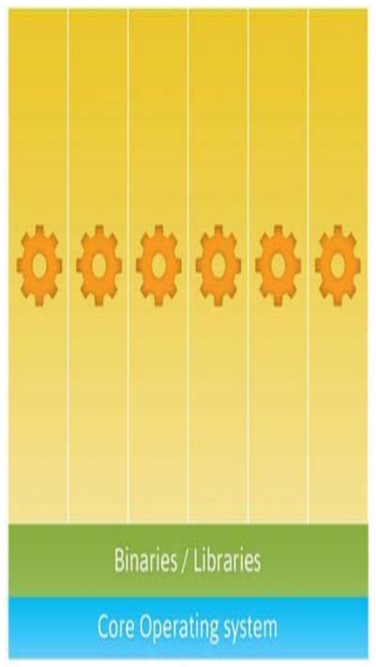
1. Easy to maintain
2. Comparative low overhead.

**Cons:**

1. Single Large Fault domain
   a. One malicious app may crush the whole system.
2. No Complete isolation.

Process Level Virtualization

## ZeroVM Virtualization



Process Level Virtualization

**Pros:**

1. Nearly Complete Isolation
   - Uses Google Native Client (**NaCl**) Project
2. Low Resource overhead.
3. Fault Tolerant

**Cons:**

1. Run Only special executables/ binary.
2.
3. No support for existing

## ZeroVM Properties

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - Channel based I/O
   - Predefine socket port / network
   - Restricted Memory Access
   - Limited Read/ Write (in bytes)
   - Limited life time / Predefined timeout

## ZeroVM Properties

**Single Threaded Execution:**

1. No Fork
2. No Context Switch
3. No Fault due to Undeterministi c concurrency

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - Channel based I/O
   - Predefine socket port / network
   - Restricted Memory Access
   - Limited Read/ Write (in bytes)
   - Limited life time / Predefined timeout

## ZeroVM Properties

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - Channel based I/O
   - Predefine socket port / network
   - Restricted Memory Access
   - Limited Read/ Write (in bytes)
   - Limited life time / Predefined timeout

## Channel Based Input / Output

Before execution ZeroVM is given a manifest/ configuration file which specify predefined Resources through Channel.

Input file, Output file / File System
Network (socket, DNS)
Memory

**Channel = /tmp/input.txt, /dev/stdin, 0, 1, 0x1000, 0x1000, 0, 0**

**Which means :**

**Zerovm input (/dev/stdin) comes from : /tmp/input.txt of local filesystem.**

**0: Only sequential Read / Write is allowed**

**0x1000: only 1000 bytes is allowed to be read from input file.**

**0: 0 bytes can be written to /tmp/input.txt**

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - Channel based I/O
   - Predefine socket port / network
   - Restricted Memory Access
   - Limited Read/ Write (in bytes)
   - Limited life time / Predefined timeout

## An example Manifest file

```
Channel = /dev/null, /dev/stdin, 0, 1, 999999, 999999, 0, 0
Channel = /dev/stdout, /dev/stdout, 0, 1, 0, 0, 999999, 999999
Channel = /dev/stderr, /dev/stderr, 0, 1, 0, 0, 999999, 999999

Version = 20130611
Program = hello.nexe
Memory = 33554432, 1
Timeout = 1
```

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - Channel based I/O
   - Predefine socket port / network
   - Restricted Memory Access
   - Limited Read/ Write (in bytes)
   - Limited life time / Predefined timeout

## ZeroVM Properties

**Single Threaded Execution:**

1. No Fork
2. No Context Switch
3. No Fault due to Undeterministic concurrency

1. ZeroVM virtualizes Application not Operating System.

2. Single threaded (thus deterministic) execution

3. Constraint Resource
   - ➢ Channel based I/O
   - ➢ Predefine socket port / network
   - ➢ Restricted Memory Access
   - ➢ Limited Read/ Write (in bytes)
   - ➢ Limited life time / Predefined timeout

## Binary Support for ZeroVM

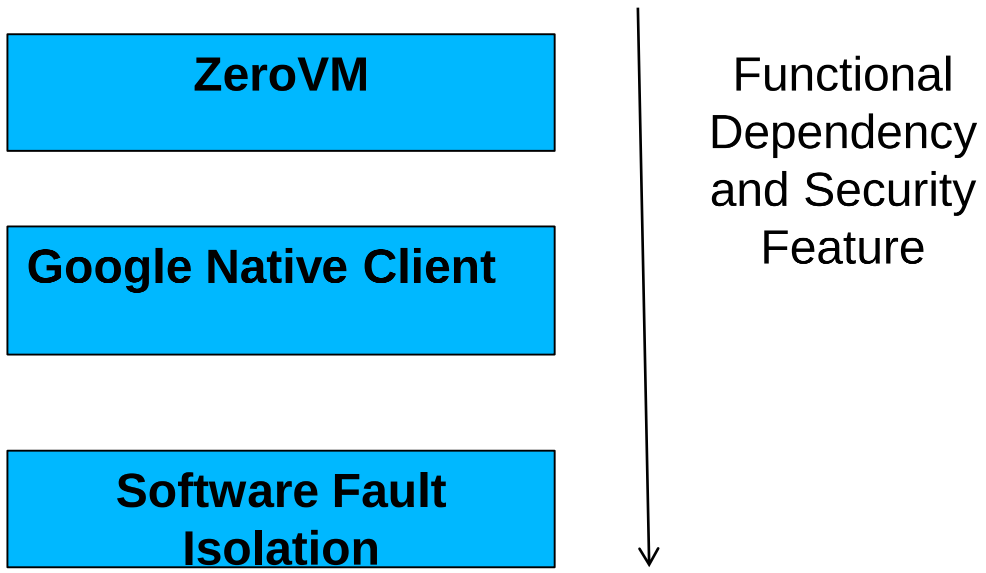
ZeroVM executables have to be precompiled in **.nexe** format.

Currently only C (C99) and python executables are supported.

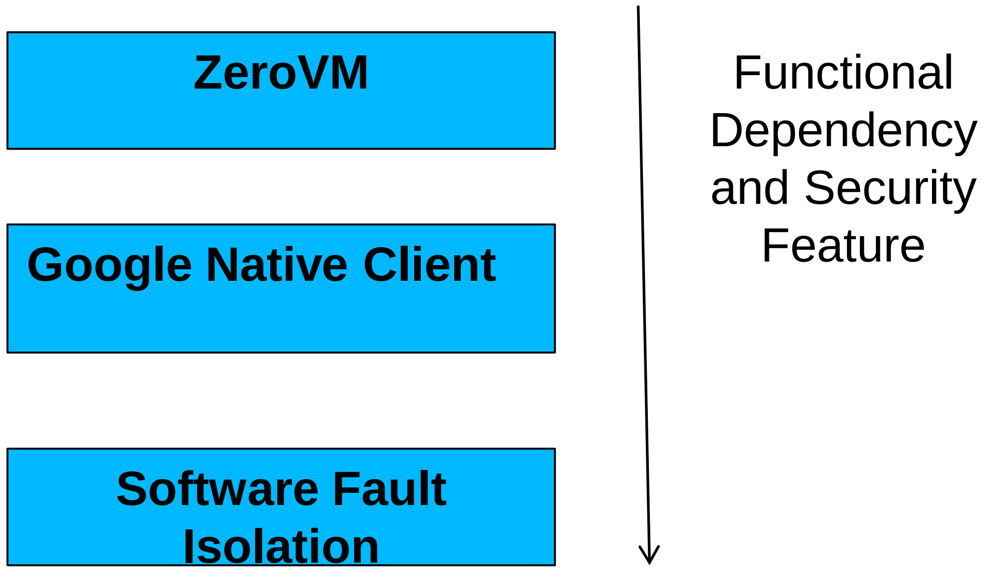
Existing C executables and python interpreter need recompilation to modify / eliminate sensitive system calls.

## ZeroVM from a theoretical standpoint

ZEROVM
the cloud hypervisor



## ZeroVM from a theoretical standpoint



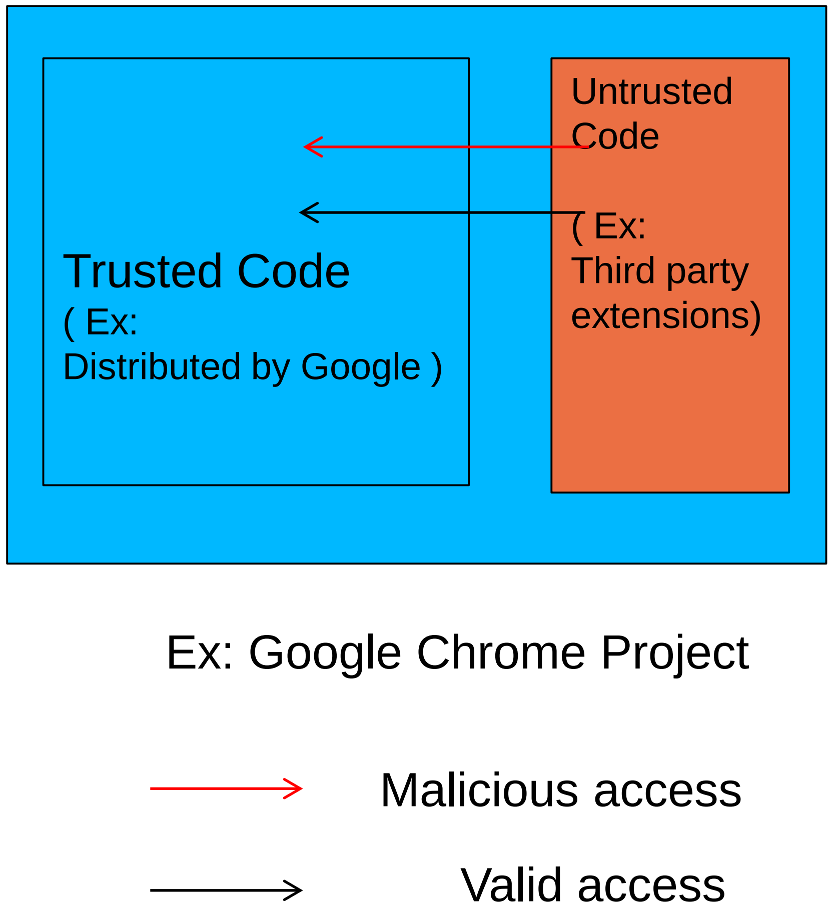
# Software Fault Isolation

**Fault Isolation Techniques:**

1. Address Space Abstraction by OS

**Cons**:

1. Communication between address space is very costly.



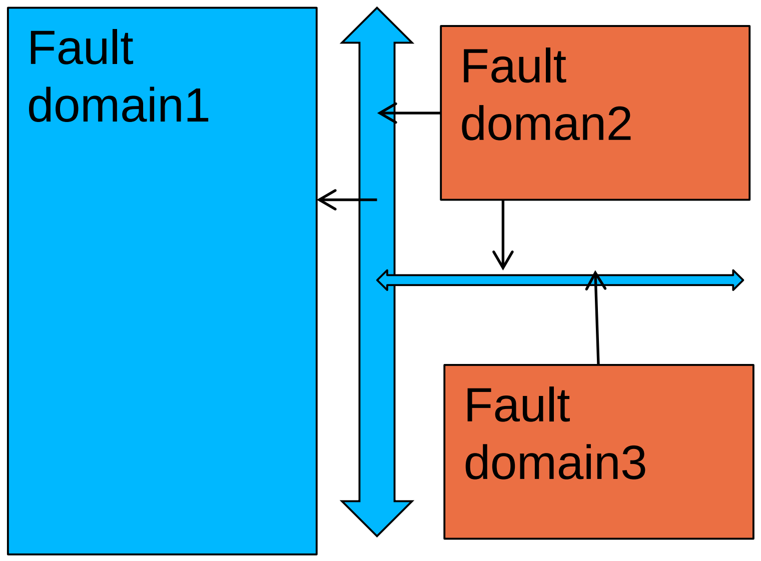
Ex: Google Chrome Project

## Software Fault Isolation

**Fault Domain:**
-- Contiguous region of memory.
-- have different code and data segment
-- Code from different trust level have own fault domain.

**Cross Domain Communication:**
**--** No direct memory access
-- All call are implemented by RPC

**Single Domain Restricted Access:**
-- the module cannot change Code segment. (dangerous, self modifying code)
-- Every jump instruction must not pass single domain.
-- Most Jumps are statically verified otherwise
-- verified at run time with help of checking code.



Distributed code / extensions must be recompiled/rewritten.

# Google Native Client (NaCl)

**Fault Domain:**

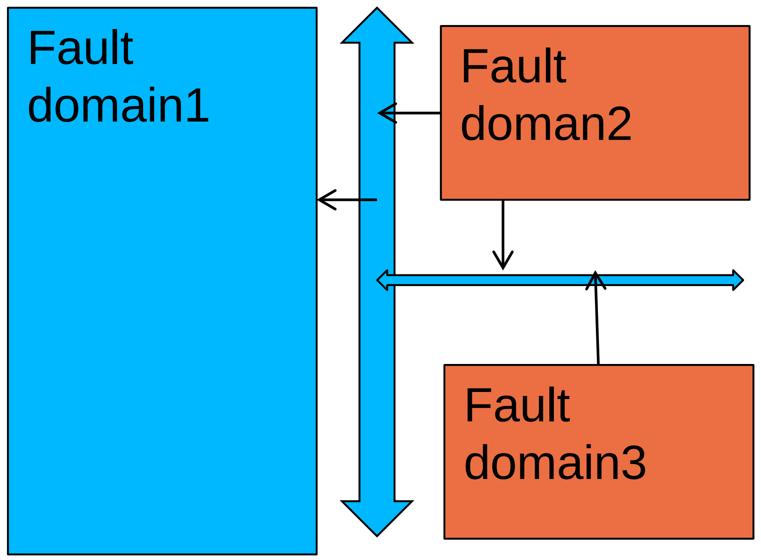
-- Contiguous region of memory.
-- have different code and data segment
-- Code from different trust level have own fault domain.

**Cross Domain Communication:**

-- No direct memory access
-- All call are implemented by RPC

**Single Domain Restricted Access:**

-- the module cannot change Code segment. (dangerous, self modifying code)
-- Every jump instruction must not pass single domain.
-- Most Jumps are statically verified otherwise
-- verified at run time with help of checking code.



Distributed code / extensions must be recompiled/rewritten.

The Institute for Cyber Security

# Thank You ☺