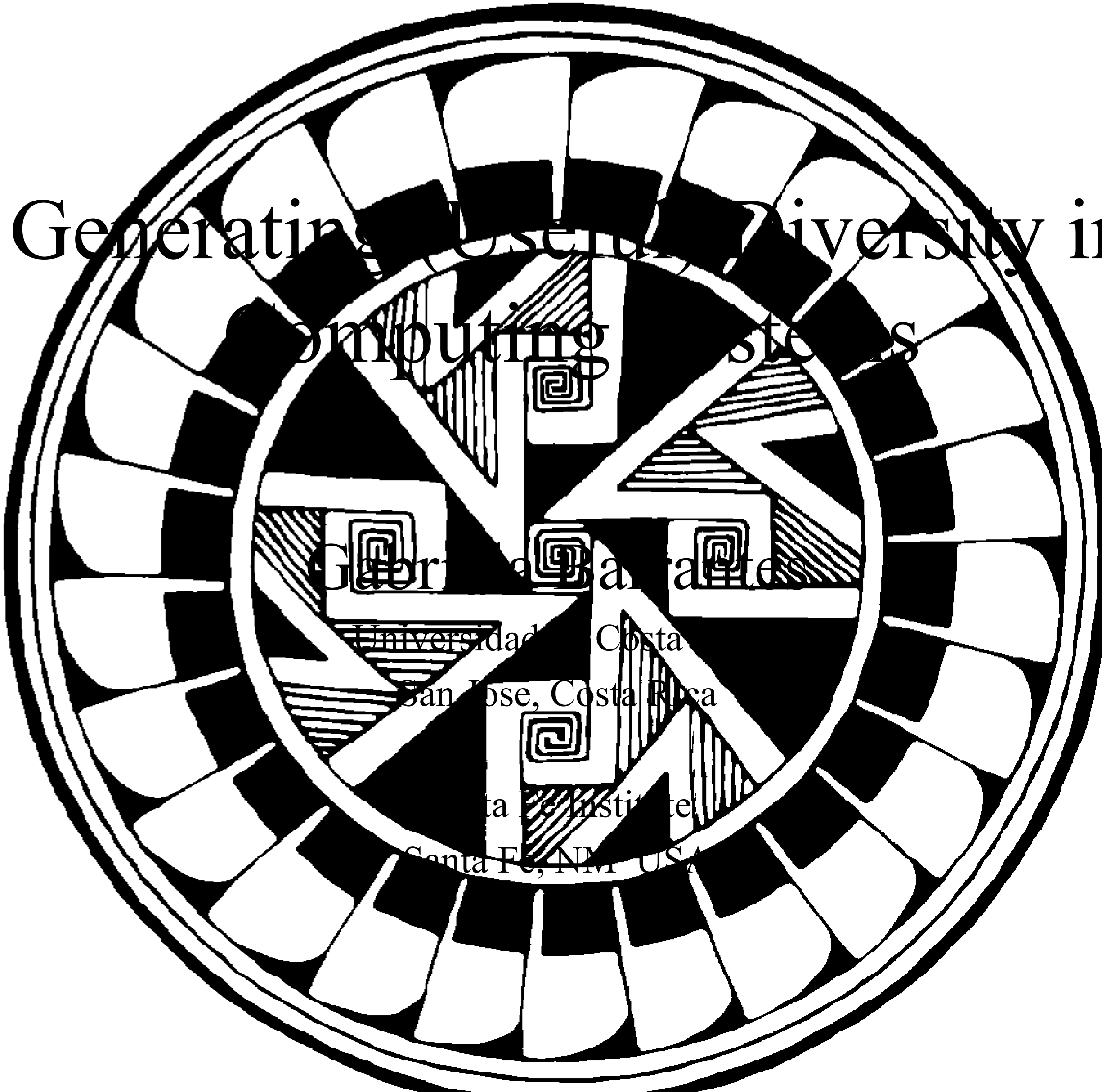


Generating (Useful) Diversity in Computing Systems

Gabriela Barrantes

Universidad de Costa Rica

San Jose, Costa Rica

Santa Fe Institute

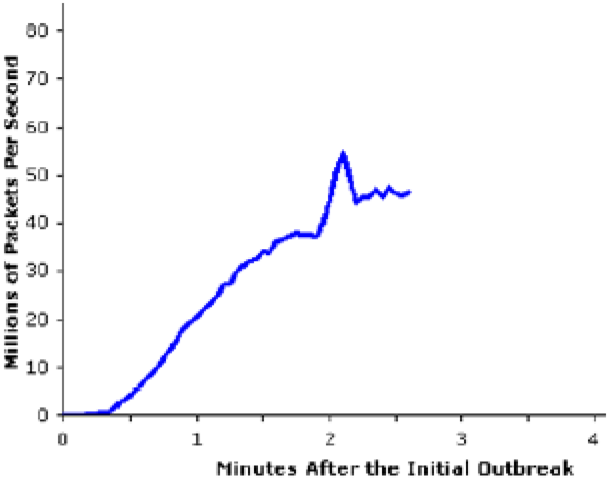
Santa Fe, NM USA

Diversity in Computer Systems

- Computer systems are becoming less diverse over time
  - Architectures (Intel, Motorola)
  - Operating systems (Unix/Linux varieties and Windows)
  - Network protocols (TCP/IP at the top)
  - Database managers (Oracle, PostgreSQL)
- Loss of diversity is even faster on higher layers
  - Web applications
  - Java

Consequences

- Ease of attack propagation
- Full connectivity
- Once an attack is successful, it will be successful on a large proportion of the network
- Example: Slammer (2003)
  - 90% of Internet was scanned in less than 10 minutes
  - A conservative estimate puts the total number of infected hosts in around 100 thousand



Savage et al. "Internet Outbreaks: Epidemiology and Defenses". Invited talk at NDSS 2005

Data from: Moore et al, IEEE Security & Privacy, 1 (4), 2003

Response solutions

- Engineered
  - Patches
  - Anti-{virus, spam, spyware}
  - Intrusion Detection Systems
  - Firewalls
  - Content filters
- Very important, but reactive
- Arms race between attackers and defenders

Proactive defenses (?)

- The biological analogy – Diversity provides population resilience to unknown environmental threats
- Introducing some automated diversity may provide resilience against attacks exploiting undiscovered vulnerabilities.
- It is possible to artificially add diversity

Dealing with the lack of diversity

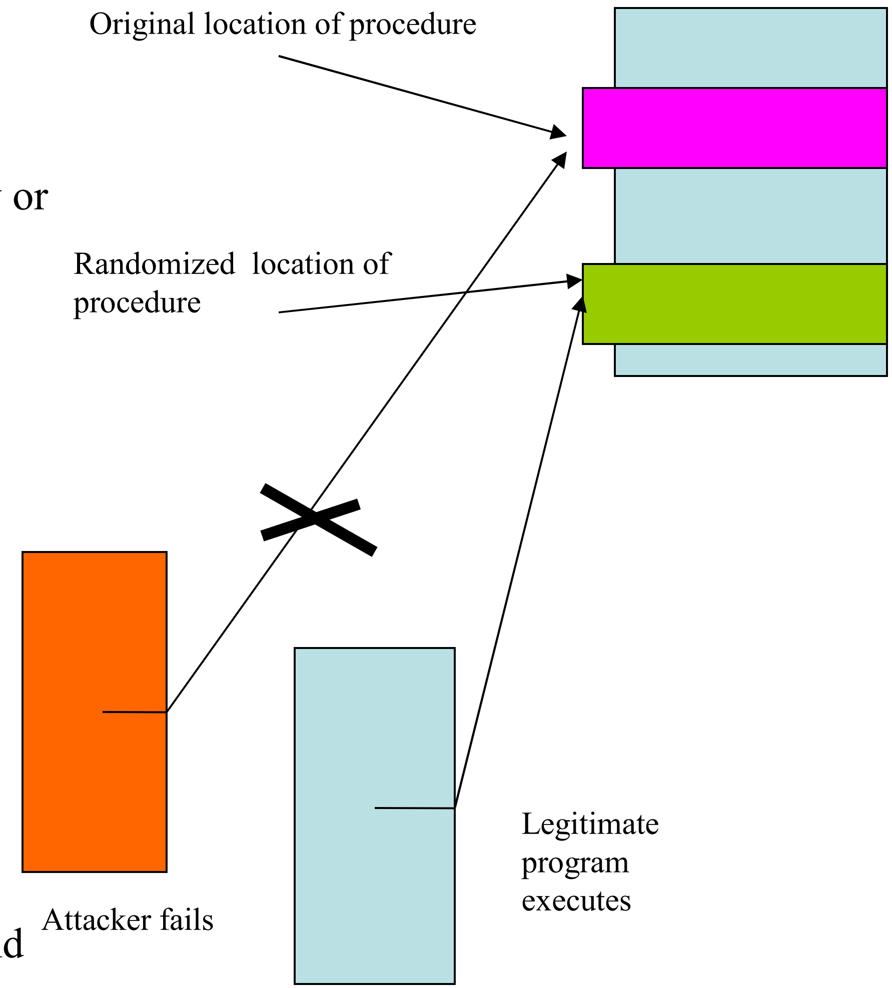
- Introducing diversity is not easy
- Uniformity provides many benefits:
  - Ease of development and maintenance
  - Interoperability
  - Compatibility
- Total diversity is infeasible.
- We can introduce it at <u>critical subsystems</u>, but we need to be very cautious:
  - Diversity is never cheap (But neither is any security measure!)
    - Implementation (one-time costs)
    - Complex or sub-optimal procedures (sustained costs)

“Critical subsystems”

- We can start by looking for invariants. Some examples:
  - Return address “below” the variables (and there is a convenient reading/writing direction)
  - A function call lists parameters two words after the function entry address
  - Heap allocations are done linearly
  - …
- Most of them should not even be there!
- Many of these invariants do not have any effect on attacks (yes, I was showing good examples). In general it is NOT easy to find places for the diversifications
- Being more organized... diversity can be located:
  - On the interface
  - On the implementation
  - On the defending systems

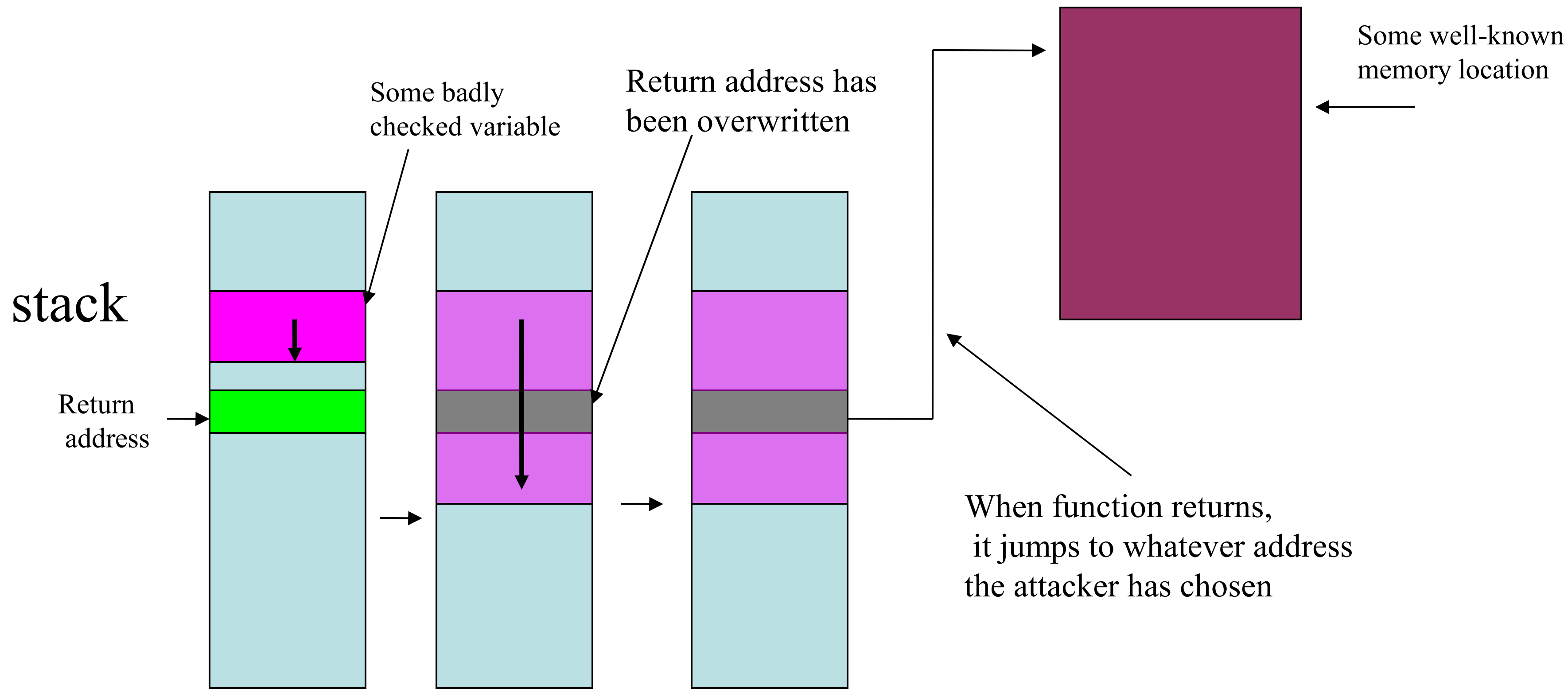
Interface diversification

- What is considered an "interface"?
  - Any arbitrary convention on object identification
  - Assignment of names or numbers to routines, "standard" locations in memory or disk, etc.
- Interface diversification is also called in literature randomization and obfuscation
- Can use encryption tools to increase the level of diversity
- Some interfaces that have been randomized:
  - Addresses
  - Machine instructions
  - System calls
  - File names
- Why does it work?
  - Attackers use standard names and (at the beginning) do not know new mapping and fail



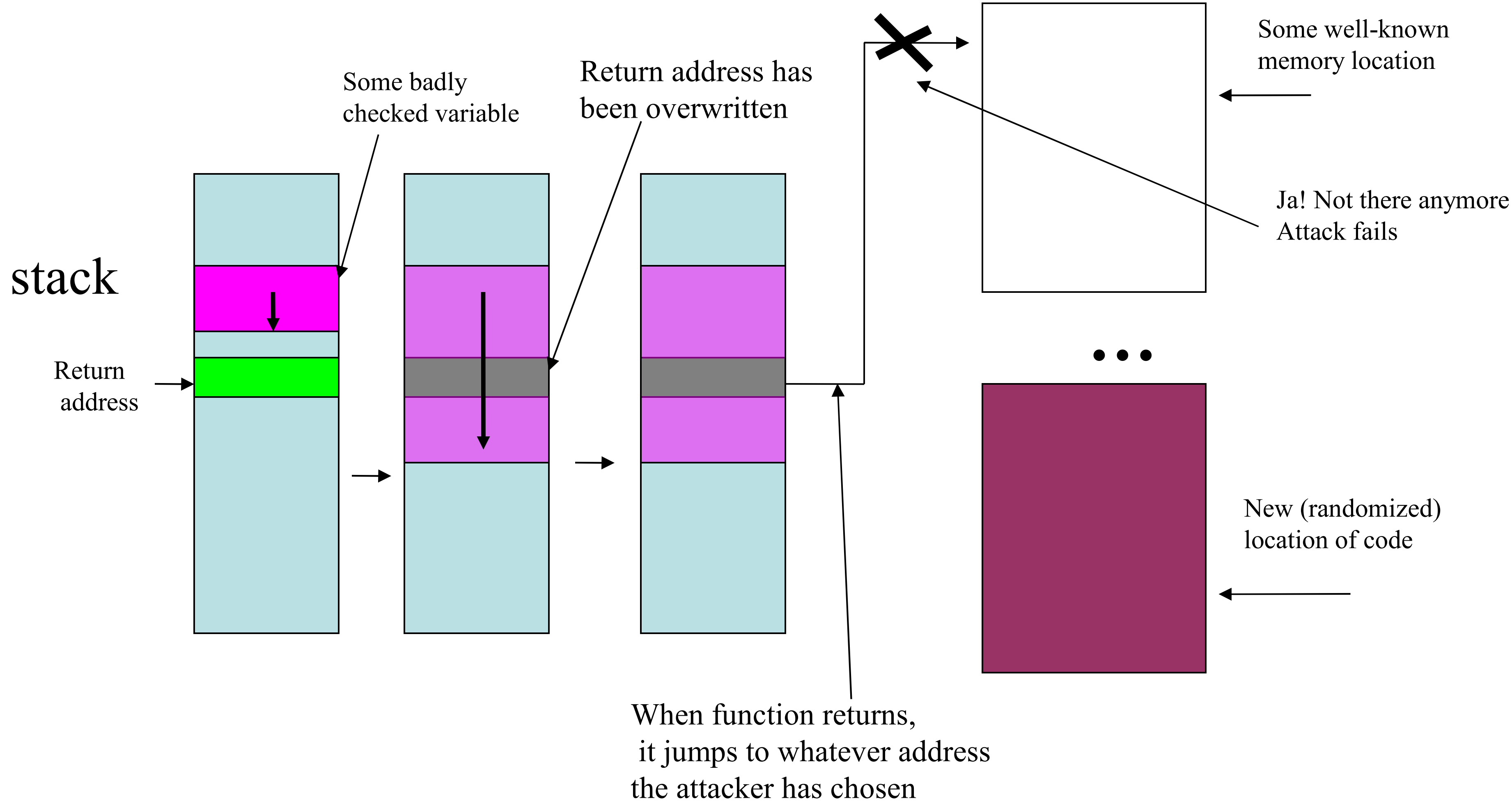
Interface randomization with addresses

- First, a brief example of how a low-level attack might operate



Interface randomization with addresses (cont'd)

- So, the randomization takes the "well-known" location elsewhere…



Interface randomization with addresses (cont'd)

- Coarse grain address diversity:
  - Programs use sets of code (libraries) that are situated in fixed locations
  - The location of the libraries and other code blocks can be randomized
  - Strategy is used in Linux (starting with PaX) and in Windows Vista
- **Fine grain address diversity**
  - The addresses of objects inside the blocks can be further randomized
    - Space between stack activation records
    - Location of heap objects
    - Location of procedures inside the block
  - Bhatkar, S. et al, 2003, 2005, 2006.

Interface randomization with Machine Instructions (ISR)

- Machine language is just another interface
- The general idea is to randomize at load and de-randomize at fetch
  - On software (using virtual machines)
    - Barrantes et al, 2003, 2005, 2006, Kc et al, 2003, Hu et al, 2006
  - On hardware (decryption at cache line level)
    - Duc et al, 2006, Wang et al, 2006.

Instruction Set Randomization example

- Diversify the interface between the machine code interpreter and the binary process.
- Use an open source emulator (Valgrind), for the IA32 architecture on the Linux OS.
- Diversification strategy:
  - **Modification of the emulator** to translate from standard IA32 machine language to the diversified language and to recognize the modified language at the time of execution.
  - **Randomization** as mechanism for creating the customized language.
- Prototype name: Randomized Instruction Set Emulation (RISE).

Threat model

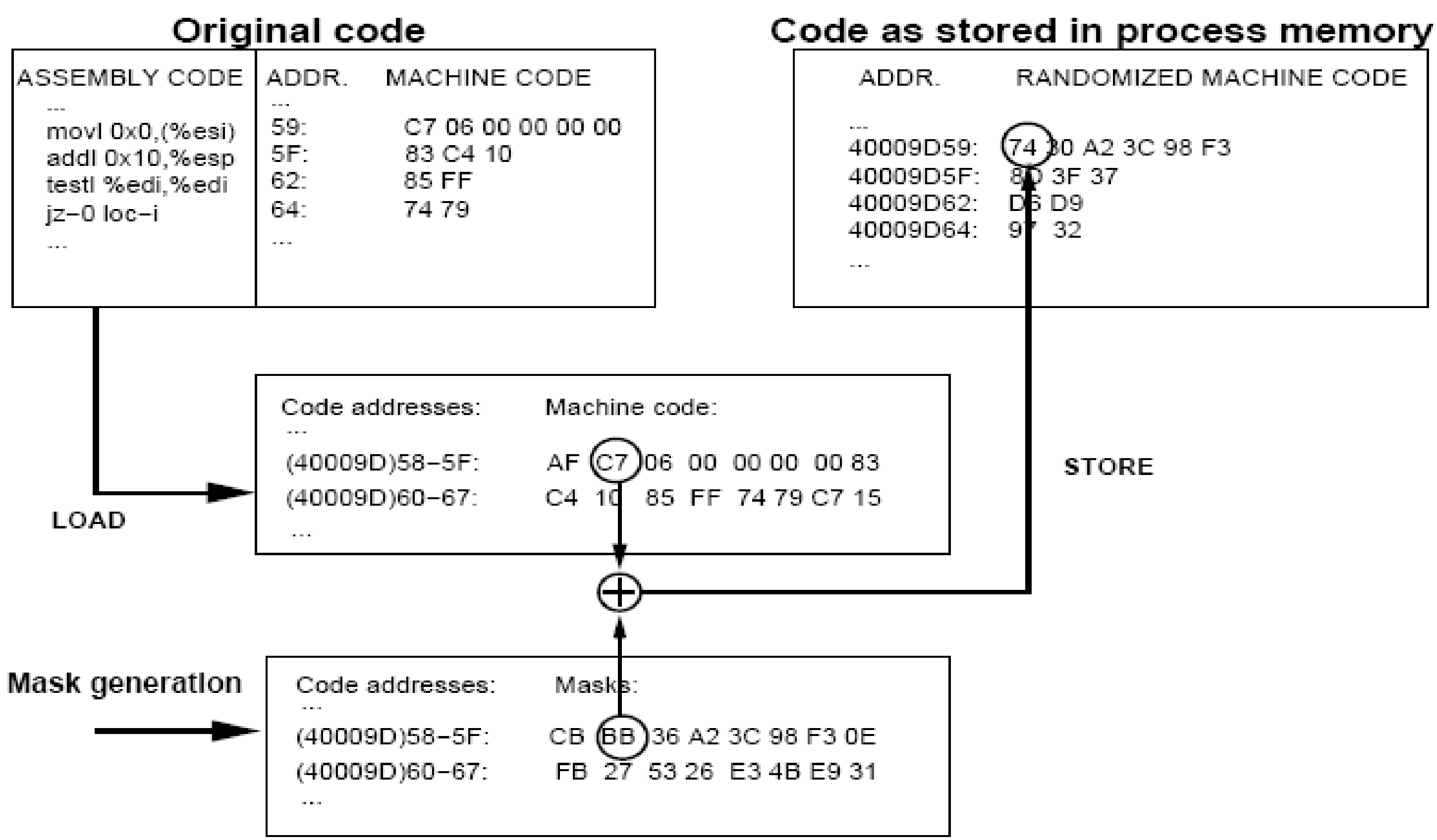
- Code injection attacks: a vulnerability is used to write ("inject") hostile code (the attack) in the target process space, with immediate or delayed execution.
- Restricted to attacks that:
  - Require execution of some machine code (binary)
  - Execute remotely and/or have limited disclosure of current process layout in memory
- Approach: Make the machine code unique to each process, so the binary attack will be expressed in the "wrong" language and fail.

RISE operation

- Two phases:
  - (a) Randomizing the executable at load:
    - Creating a unique language
  - (b) De-randomizing instructions at fetch:
    - Interpreting the new language correctly

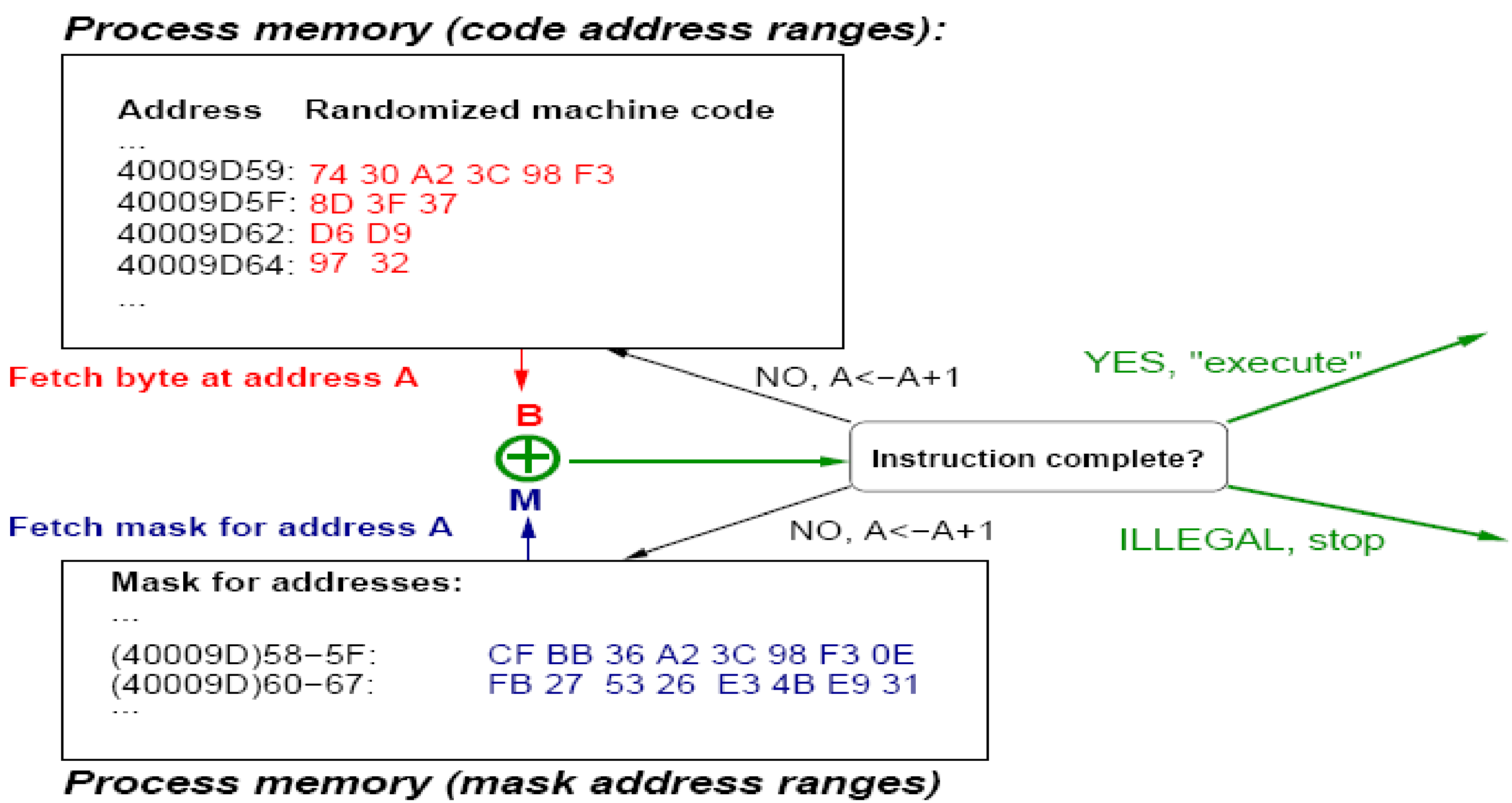
RISE operation (cont'd)

(a) Creating a unique language

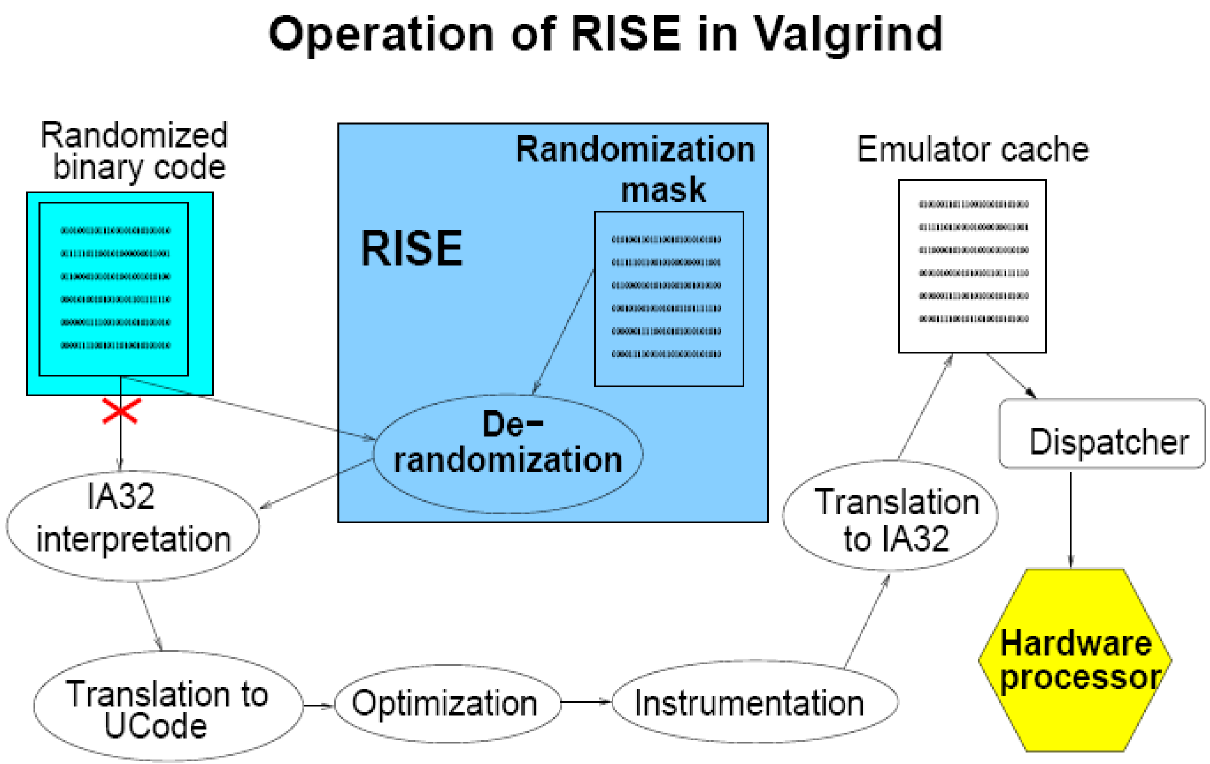


RISE operation (cont'd)

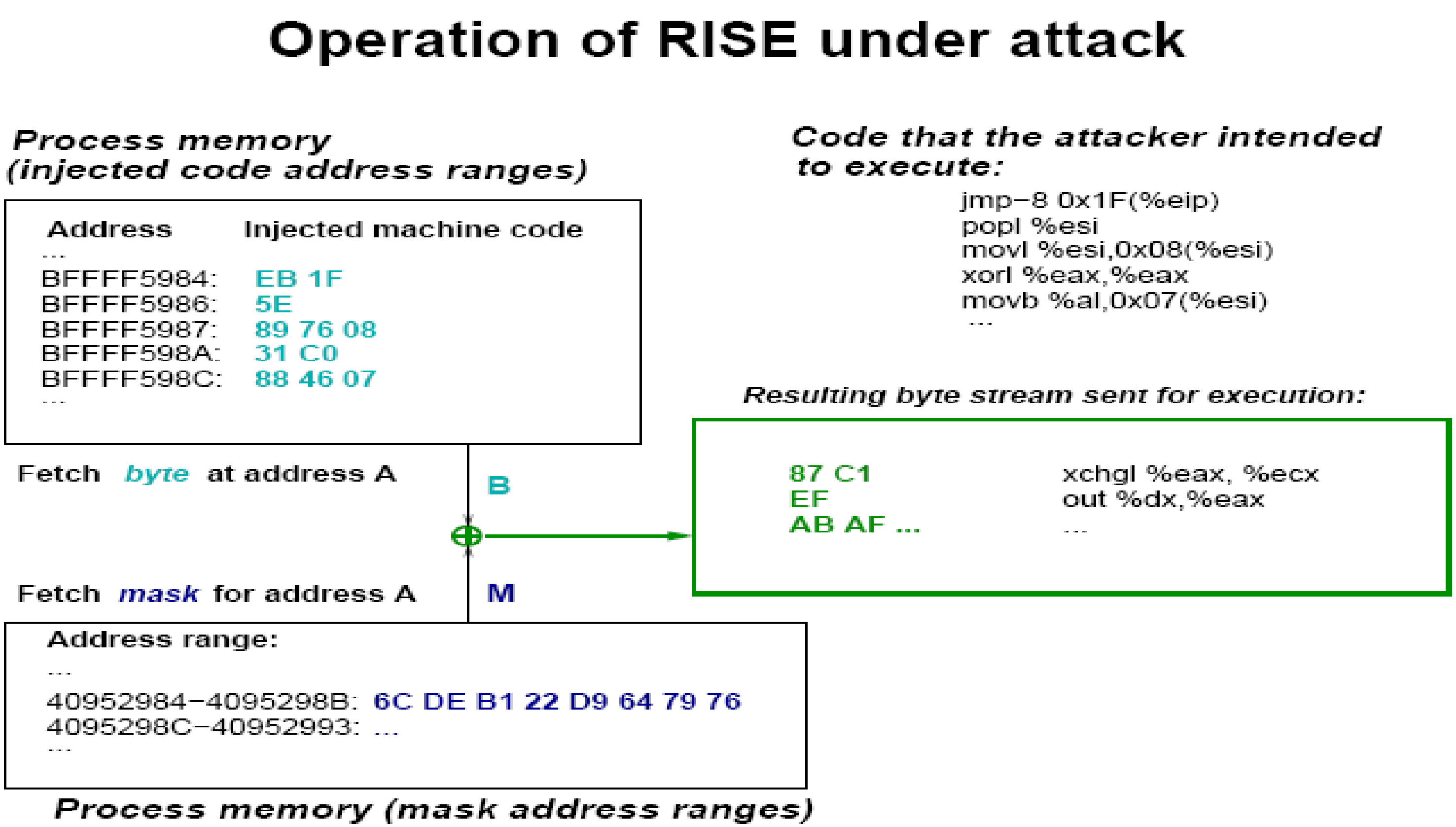
(b) Interpreting the new language



RISE operation (cont'd)



RISE operation (cont'd)



Effectiveness of RISE against attacks

Select attacks in threat model from CORE Impact penetration tool. Vulnerable applications running under RISE.

Attack	Linux Distribution	Vulnerability	Location of injected code	Stopped by RISE
Apache OpenSSL SSLv2	RedHat 7.0 & 7.2	Buffer Overflow & malloc/free	Heap	√
Apache mod php	RedHat 7.2	Buffer Overflow	Heap	√
Bind NXT	RedHat 6.2	Buffer Overflow	Stack	√
Bind TSIG	RedHat 6.2	Buffer Overflow	Stack	√
CVS flag insertion heap exploit	RedHat 7.2 & 7.3	malloc/free	Heap	√
CVS pserver double free	RedHat 7.3	malloc/free	Heap	√
PoPToP Negative Read	RedHat 9	Integer error	Heap	√
ProFTPD _xlate_ascii _write off-by-two	RedHat 9	Buffer overflow	Heap	√
rpc.statd format string	RedHat 6.2	Format string	GOT	√
SAMBA nttrans	RedHat 7.2	Buffer overflow	Heap	√
SAMBA trans2	RedHat 7.2	Buffer overflow	Stack	√
SSH integer overflow	Mandrake 7.2	Integer error	Stack	√
sendmail crackaddr	RedHat 7.3	Buffer overflow	Heap	√
wuftpd format string	RedHat 6.2–7.3	Format string	Stack	√
wuftpd glob "~{"	RedHat 6.2–7.3	Buffer overflow	Heap	√

Implementation issues

Randomization mechanism The use of an XOR-based data-hiding was chosen because it is not block-based and is very cheap to implement. There is one different byte of mask for each address used for code. Randomness is obtained via `/dev/urandom` after guaranteeing a true random seed of at least 256 bytes in `/dev/random`.

Shared libraries Every process protected by RISE has to carry its own copies of any shared libraries it uses.

Lessons learned

- Change in interface must have consequences.
- Language must have a clear boundary between trusted and untrusted applications.
- Randomization secret must be difficult to discover and/or frequently changed.
- Conceptually simple, implementation issues make it complex.

Does it work?

- Parts of it are already being used in current OSs
- All randomizations mentioned work against low-level attacks that rely on knowledge about the memory layout, but there are more interesting problems!
- Makes life <u>a little bit more difficult</u> (not impossible) for the attacker (Shacham et al, 2004, Sovarel et al, 2005, Barrantes et al, 2006)
- The problem is that most diversifications are interface-based, and therefore rely on an obfuscation key, that can be:
  - Stolen
  - Guessed (brute force must never be underestimated)

Characterizing diversity

- More serious effort needed to characterize the effect –of diversity - some work already being done in this area:
  - Characterization of propagation
  - Effect on non-naïve attacks
  - Performance vs. defense capabilities
  - …
- And of course, how to measure “diversity”?
  - It is impossible to completely obscure code and NO general obfuscator is possible (Barak, 2001)
  - Statistical Independence? (Littlewood et al., 2004)
  - Enforcing differences? (O'Donnell et al., 2004)
  - Epidemiological models? (O'Donnell et al., 2005)

What about higher layers?

- Serious problem… it is more difficult to randomize the interface – compatibility issues are harder
  - Perl and SQL interpreters with random tags (Kc et al, 2003, Boyd et al, 2004)
- Need diversity at another level: implementation
  - Similar to n-version diversity: creating diverse implementations of a program in order for some individuals to survive
- Examples
  - N-variant systems (Cox et al, 2006)
  - Policy diversification in multi-agent systems (Paruchuri et al, 2006)
  - Node ID randomization in sensor networks (Alarifi et al, 2006)
  - TCP parameter randomization (Barrantes et al, 2006)
  - Adaptive filter generation against DoS (Barrantes et al, unpublished)

What about the atttackers?

- There is evidence that attacks are being diversified (Ma et al, 2006)
- Unexpectedly…
  - Diversification is manual and not directed at avoiding signature scanners
  - Seems that there is not enough evolutionary pressure
- As cost of attack goes up, we predict that attackers will start increasing the diversification level
- For now, it is just too easy out there…

Summary

- Computer systems are too homogeneous
- Artificial diversification is necessary, and it is being used
- It helps but it is not a magical bullet
- Not very well understood
- Most potential is on implementation diversification

Acknowledgements

- ISCV
- Santa Fe Institute
- Universidad de Costa Rica
- University of New Mexico