On Evolving Buffer Overflow Attacks Using Genetic Programming

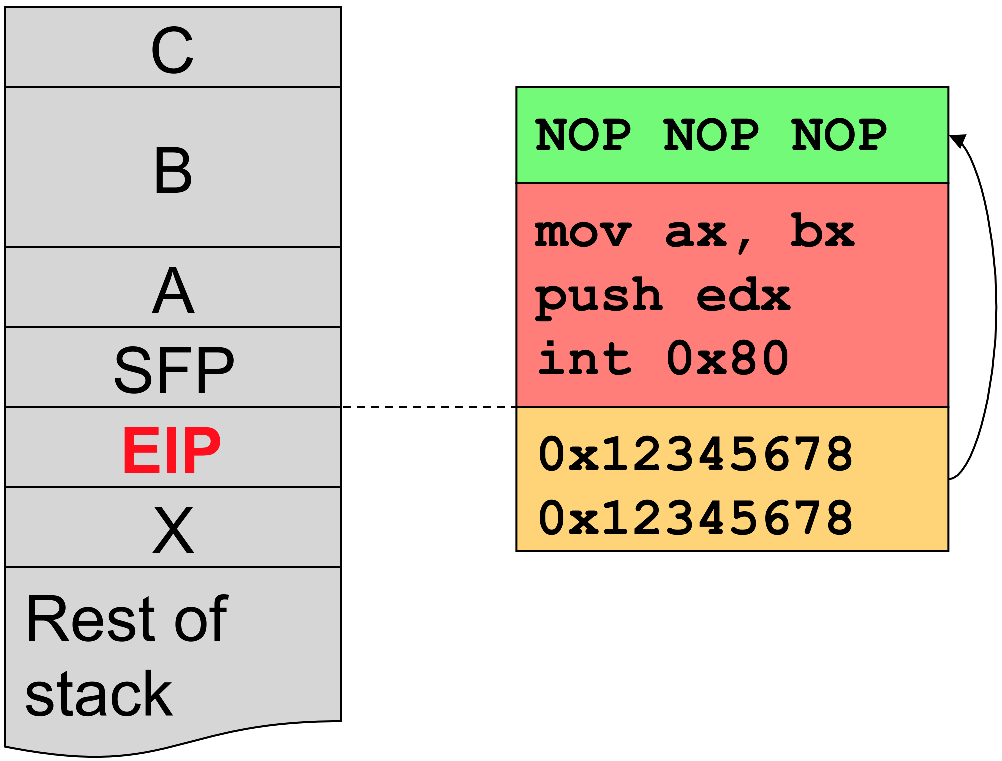
Hilmi Güneş Kayacık
Malcolm Heywood
Nur Zincir-Heywood

Introduction

- Detectors rely on a ‘third party’ to identify the new attacks.
- Objective: To evolve a ‘white hat’ attacker.
- Use code bloat property to hide the true intent.
- A detector will be built on the generated attacks.

Stack Overflow Example

Stack Layout **Malicious Buffer**



Vulnerable App.

```
Fun(char *X)
{
   char A[4];
   char B[8];
   char C[4];
   strcpy(B,X);
}
```

Spawning a UNIX shell

```
int execve(const char *path, char *const
    argv[], char *const envp[])
```

a) Register EAX contains 0x0B i.e., the system call number of 'execve';
b) Register EBX points to '/bin/sh0' on the stack;
c) Register ECX points to the argument array in stack;
d) Register EDX contains NULL;
e) Interrupt '0x80' is executed;

Linear GP

- As opposed to tree based.
- Individual is assembly code
- Instructions that are composed from a 2 byte opcode and two operands (1 byte).
- Fixed length individuals.

Fitness Function

Fitness= 10

Objective	# instructions
a. Stack contains “/bin/sh”?	1 to 3
b. EBX points to (a) ?	1
c. ECX points to arguments?	1 to 3
d. Is EDX null?	1
e. Interrupt executed?	1

Training Parameters

Parameter	Setting (Probability)
Crossover	Page Based (0.9)
Mutation	Uniform instruction wide (0.5)
Swap	Instruction swap within an individual (0.5)
Selection	Tournament
Stop	At the end of 50,000 tournaments
Population	500 individuals each with 10 pages, 3 instructions per page

Experiments

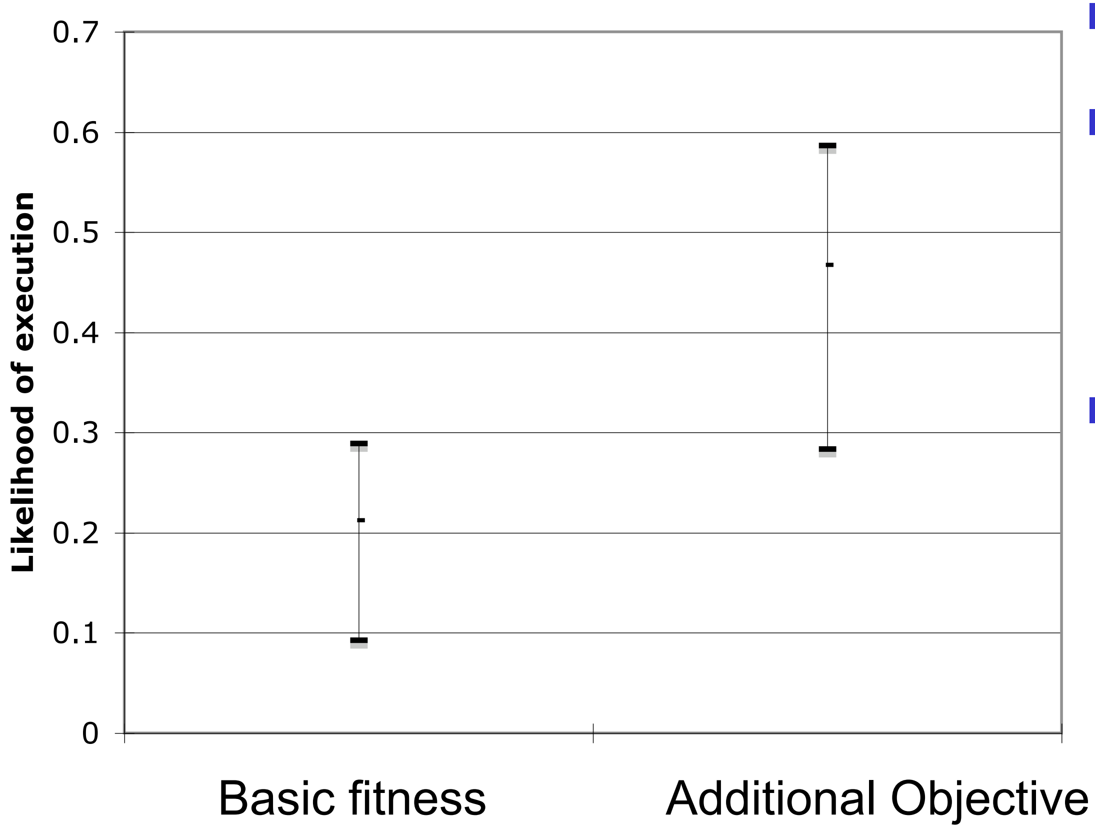
- Minimal Instruction Set
  - 5 instructions to build the attack
  - Establish a baseline
  - Additional objective to “strengthen” the attacks
- Extended Instruction Sets
  - Add arithmetic instructions
  - Add logic instructions

Instruction Set

R:	Register
I:	Immediate

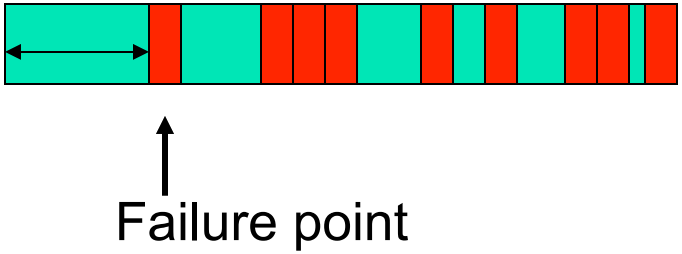
- `CDQ`
- `PUSH I`
- `PUSH R`
- `MOV R, R`
- `MOV R, I`
- `XOR R, R`
- `ADD R, R`
- `SUB R, R`
- `INC R`
- `DEC R`
- `MUL R`
- `DIV R`
- `AND R, R`
- `OR R, R`
- `NOT R`

Likelihood of Execution

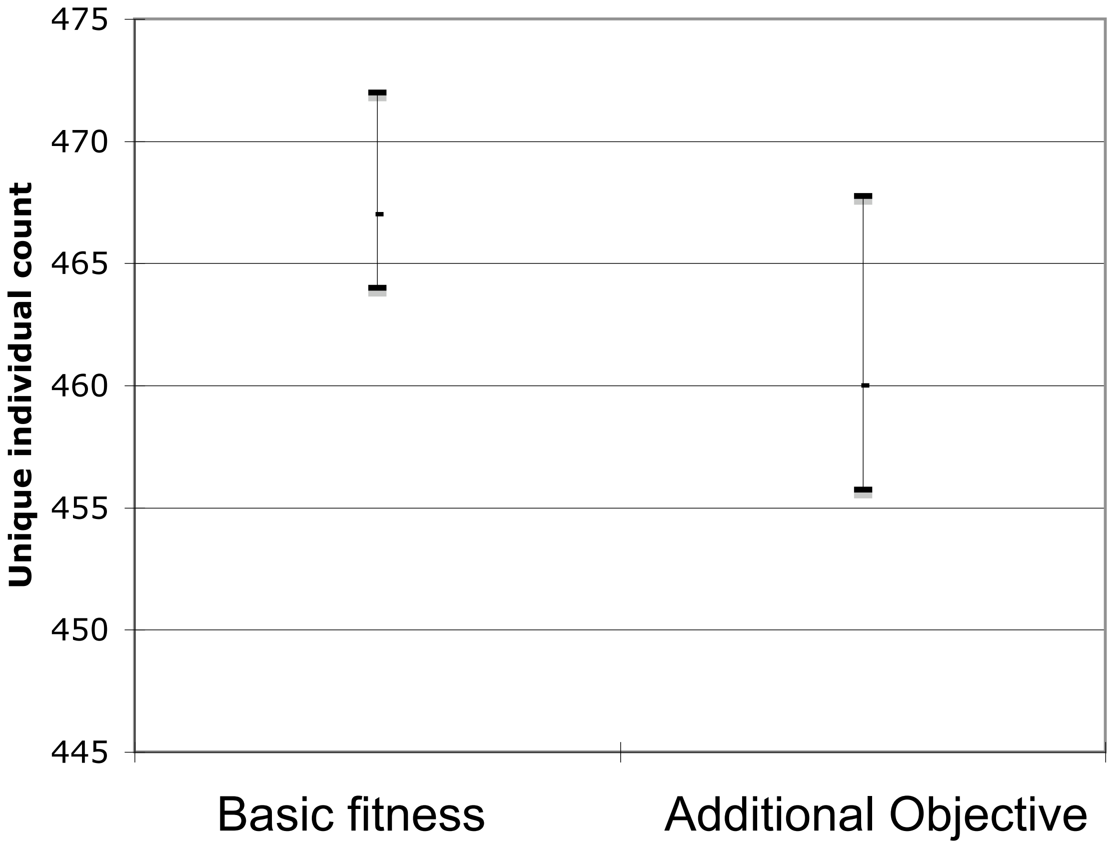


- Jump Accuracy
- Jumping to 1st attack instruction

  vs.
- Jumping to an intron

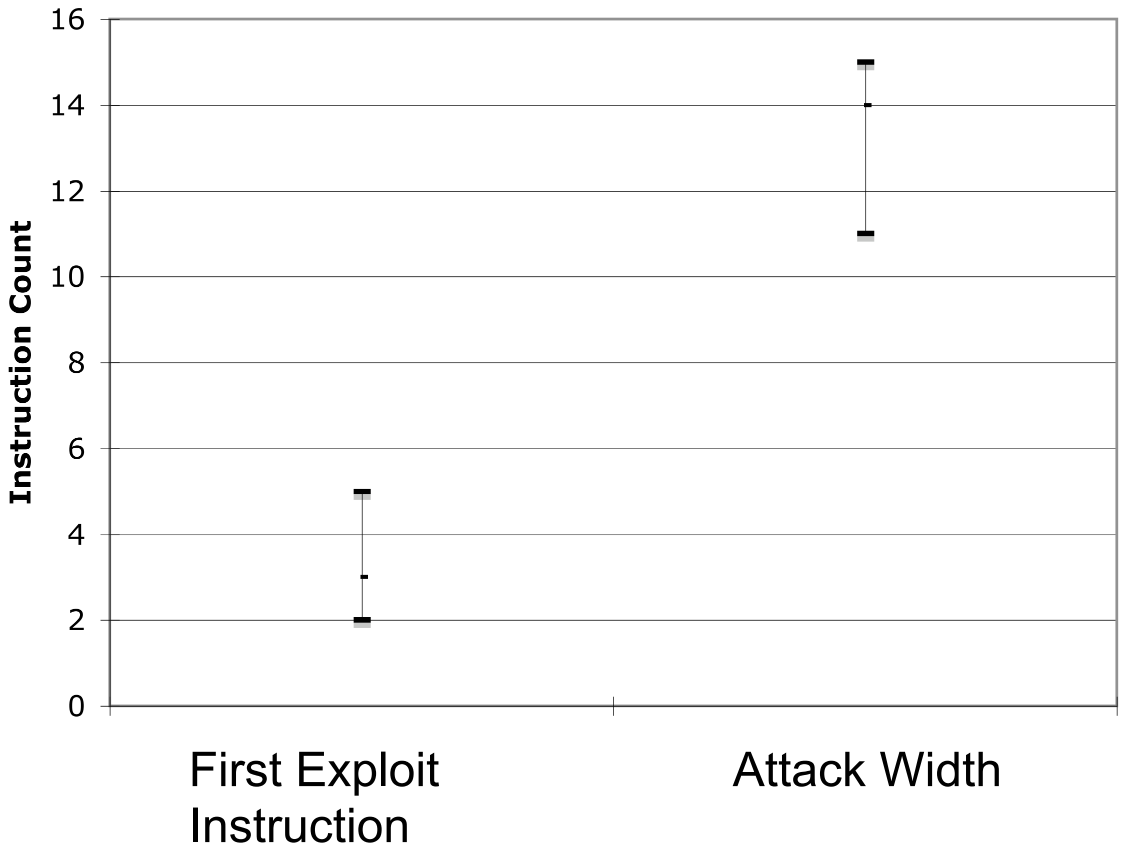


Unique Individual Count



- Unique Individual: Differs from others by at least one or more instruction

Intron Characteristics

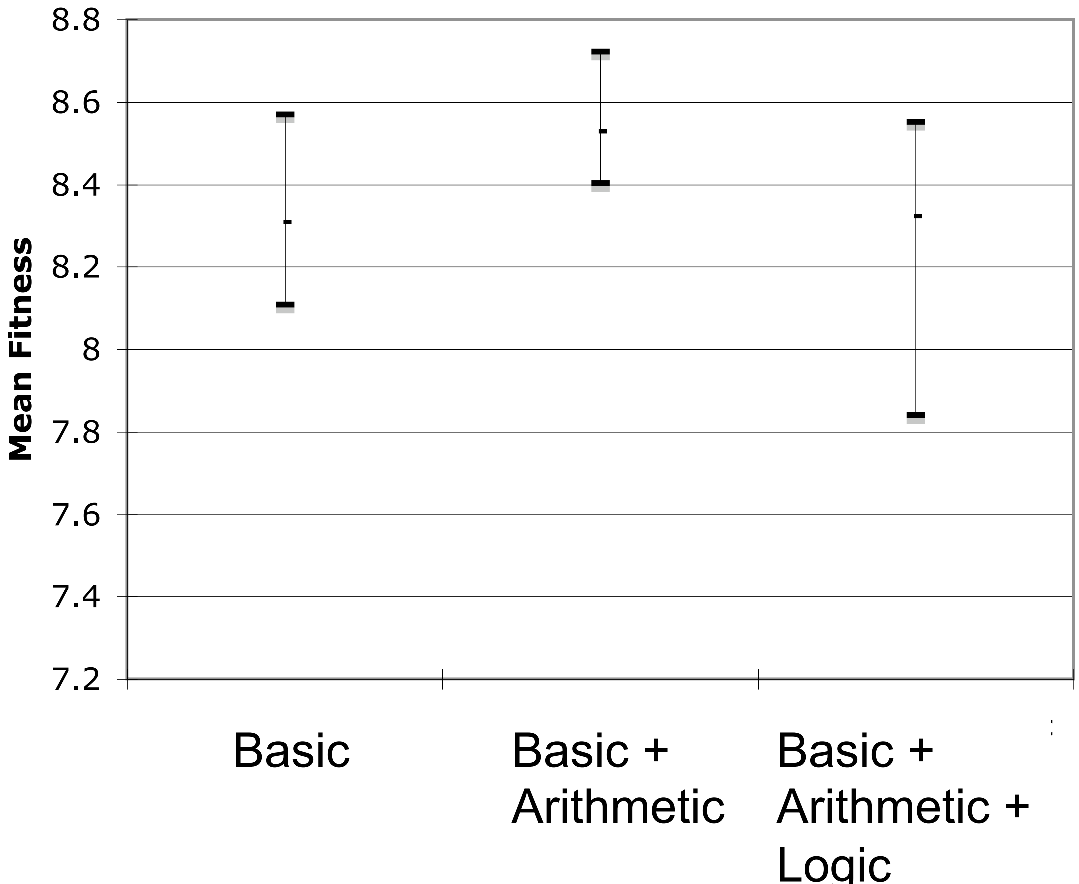


- Attack starts in the first third of the code.
- Introns are mixed with attack instructions

Comparison Between Evolved and Core Attack

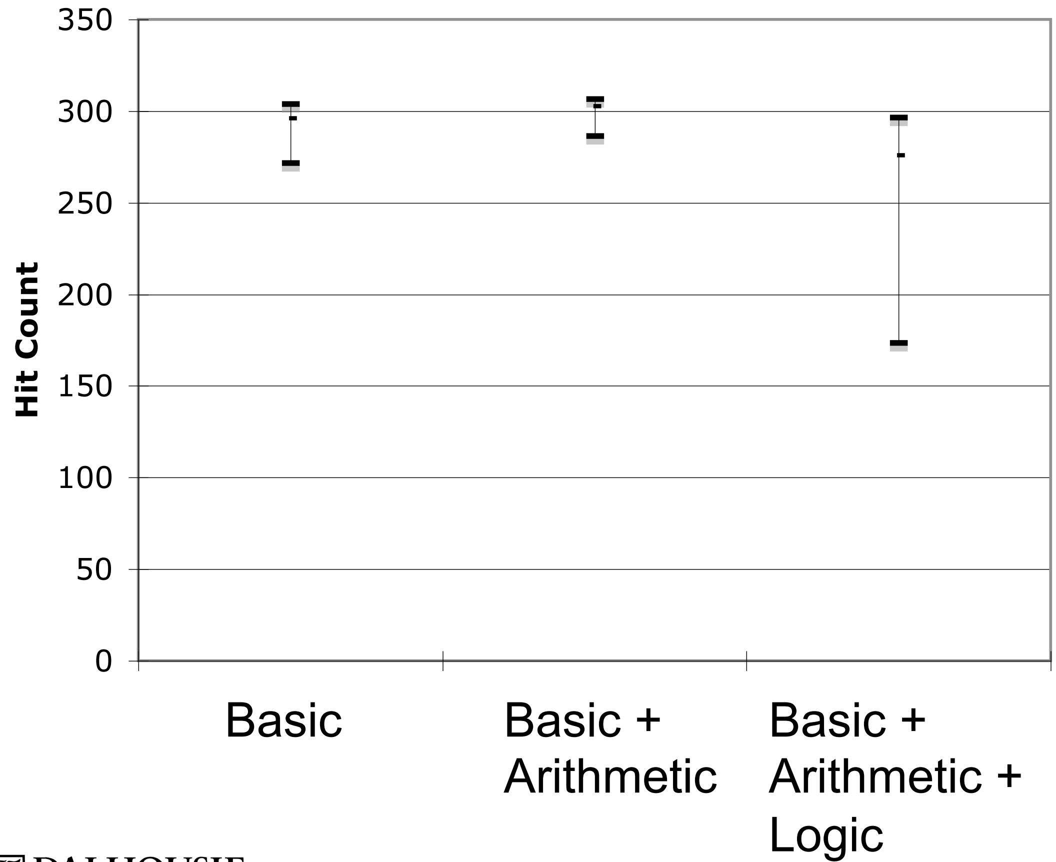
Evolved Program	Core Attack	Sub-goals
PUSH 0x68732f2f		
MUL EAX		
PUSH EBX		
MUL EDX		
CDQ		
CDQ		
SUB EAX, EAX	XOR EAX, EAX	(d)
MUL EDX	CDQ	(d)
PUSH EDX		
MOV CL, 0x0b		
PUSH EDX		
DEC ECX		
DEC ECX		
MOV EBX, ESP		
PUSH 0x6e69622f		
PUSH EDX	PUSH EAX	(a)
PUSH 0x68732f2f	Same	(a)
PUSH 0x6e69622f	Same	(a)
MOV EBX, ESP	Same	(b)
MOV ECX, EDX	PUSH EAX (step 1)	(c)
CDQ		
MUL EDX		
PUSH ECX	PUSH EAX (step 2)	(c)
PUSH EBX	Same	(c)
MOV ECX, ESP	Same	(c)
MOV AL, 0x0b	Same	(e)
INT 0x80	Same	(e)
PUSH EDX		
PUSH 0x6e69622f		
MOV DL, 0x0b		

Mean Fitness



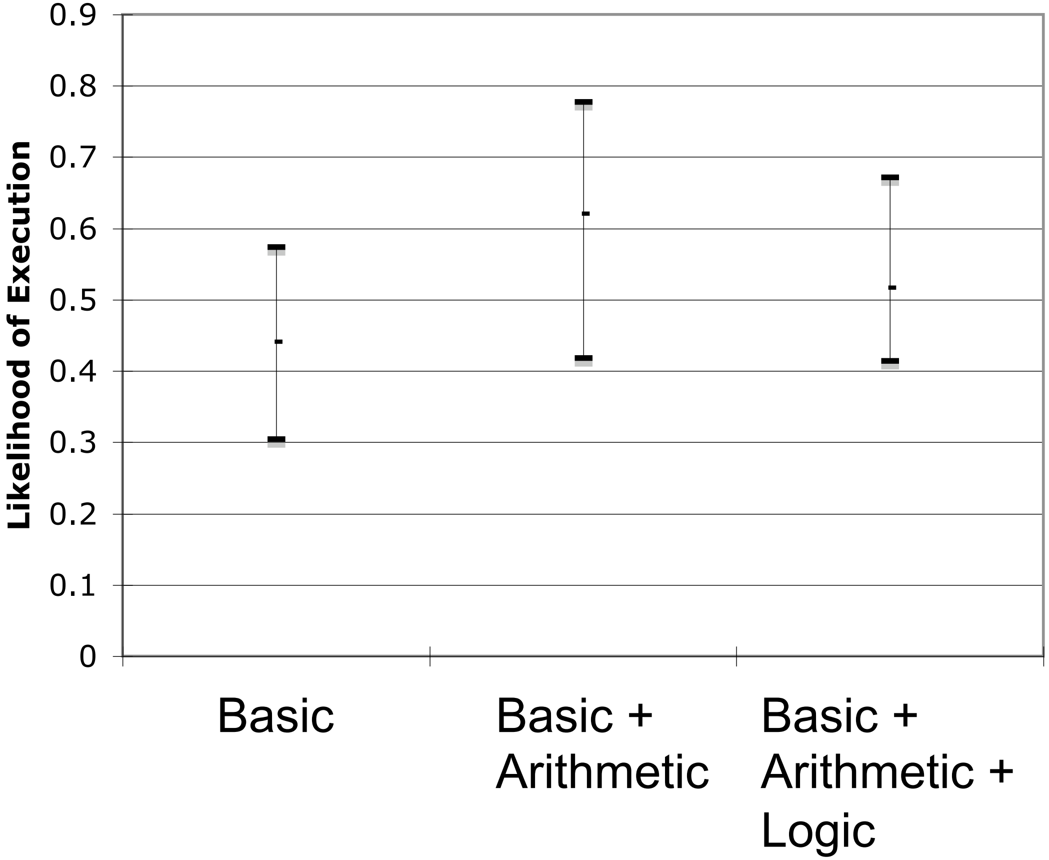
- Three instruction sets:
  1. Basic
  2. (1) + Arithmetic
  3. (2) + Logical

Hit Count

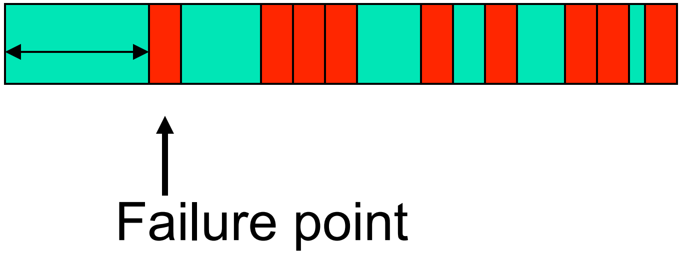


- Hit = Attack deploys successfully

Likelihood of Execution



- Jumping to an intron



Conclusions

- A ‘white hat’ attacker altering the core attack to make it undetectable.
- Code bloat provides means to hide true intent.
- Attackers discover different ways to achieve objectives.
- Experiments on fitness function and instruction sets.

Conclusions & Future Work

- Discussion of results
  - Employing additional fitness objective.
  - Expanding the instruction set.
- Future work
  - Worms and other buffer overflows
  - Coevolutionary framework between attackers and detectors.

Acknowledgements

- This work was supported in part by Killam, NSERC, MITACS and CFI.
- All research was conducted at the NIMS Laboratory, http://www.cs.dal.ca/projectx/