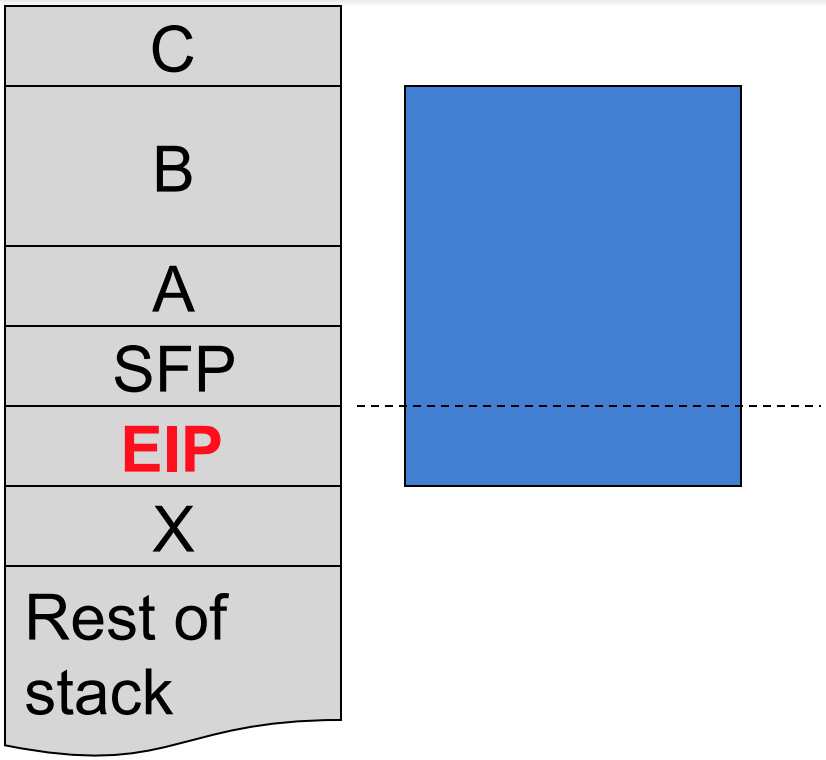
Evolving Successful Stack Overflow Attacks for Vulnerability Testing

H. Güneş Kayacık
A. Nur Zincir-Heywood
Malcolm Heywood

Motivation

- Buffer overflow attacks.
- Arms race between attackers and defenders.
- Use evolutionary computation to discover ‘good’ malicious variables.
- Use Snort to observe detection.

Anatomy of an Overflow



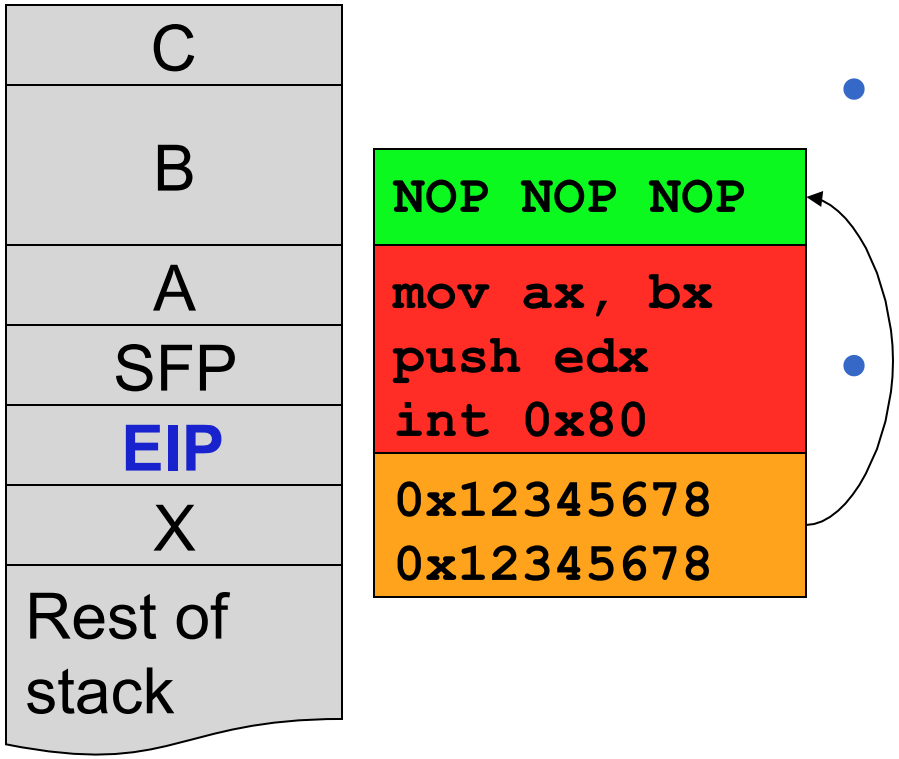

```
Fun(char *X)
{
   char A[4];
   char B[8];
   char C[4];
   strcpy(B,X);
}
```

- How stack is used

Attack Components

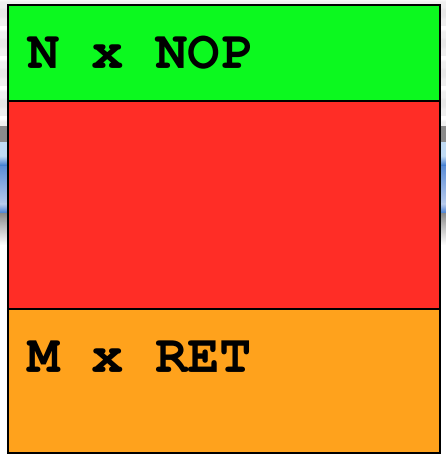
- Task: Alter the execution (i.e. modify EIP)

  To where? To the beginning of your shellcode.



- Challenge: Where does it begin **exactly**?
- Fudge Factor: NOP sled.

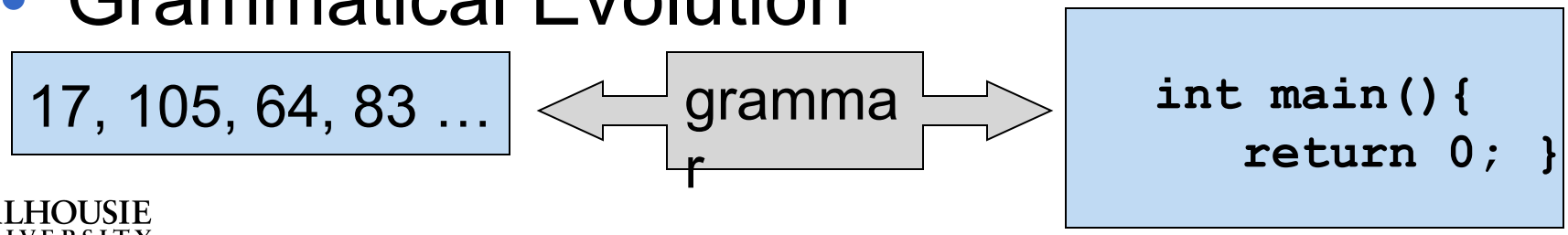
Methodology



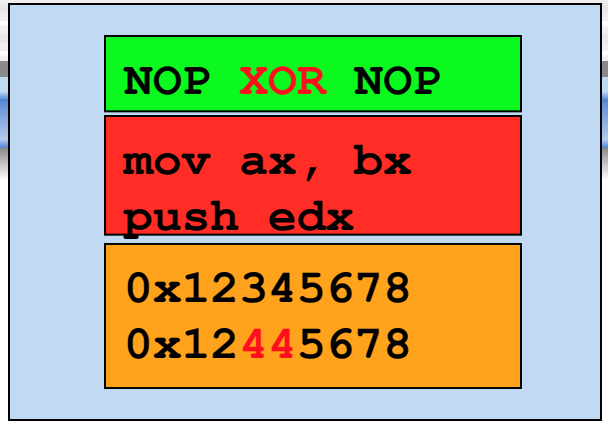
- "Evolve" programs that will:
  - Approximate return address (**RET**)
  - Determine number of return addresses (**M**)
  - Determine NOP sled size (**N**)
  - Finally, assemble the malicious buffer.
- Vulnerable application
- Snort IDS
- Further experiments to encourage diversity.

Grammatical Evolution

- Based on
  - Population of solutions (or individuals)
  - Survival of the fittest
  - Fitness function
  - Search operators
    - Mutation
    - Crossover
- Grammatical Evolution



Fitness Function



Shellcode exists?

N → 0

Y

Does it work?

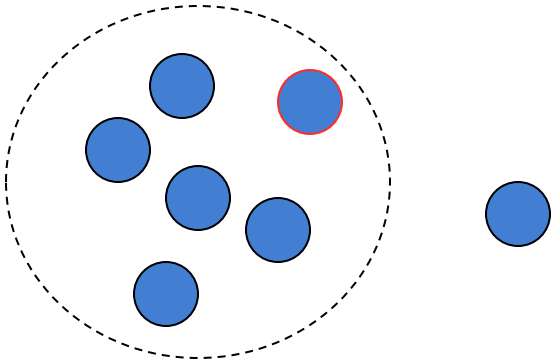
N → $NOP_{error} + RET_{error} + RET_{accuracy}$

Y

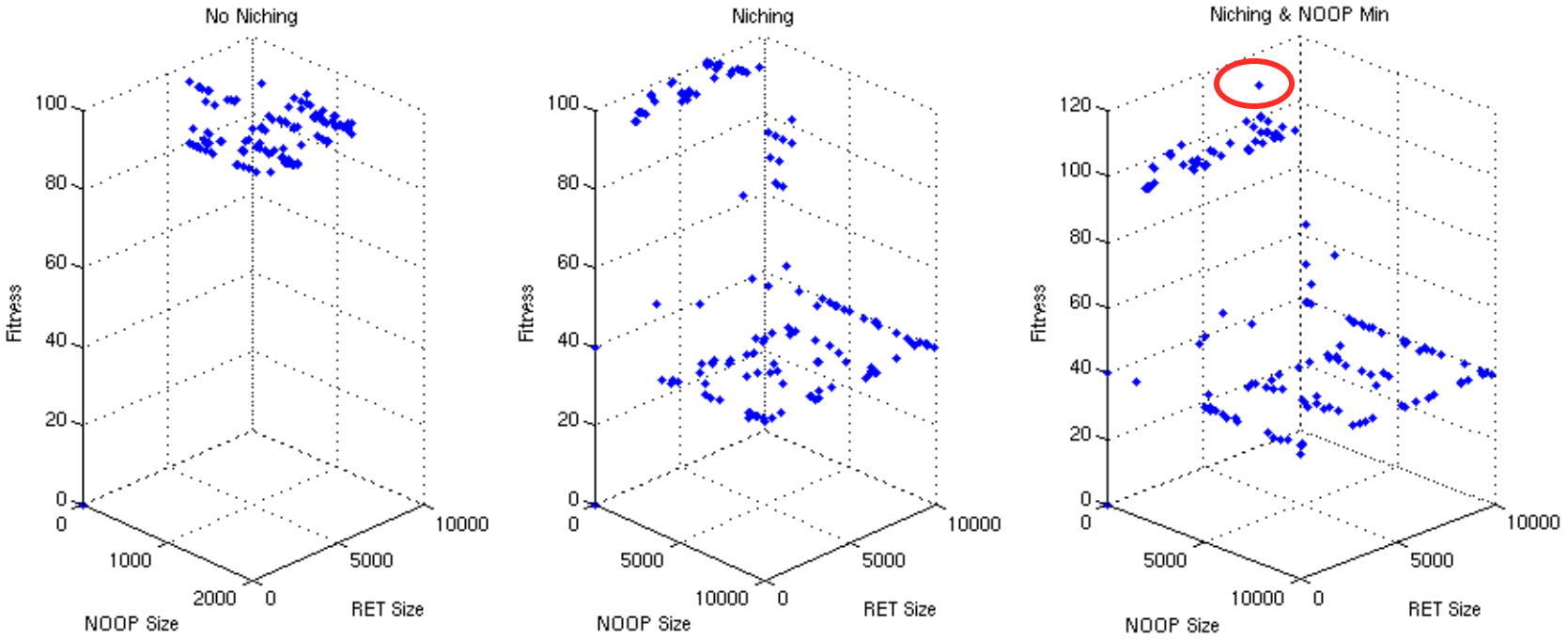
$100 + NOP_{score}$

Fitness Sharing

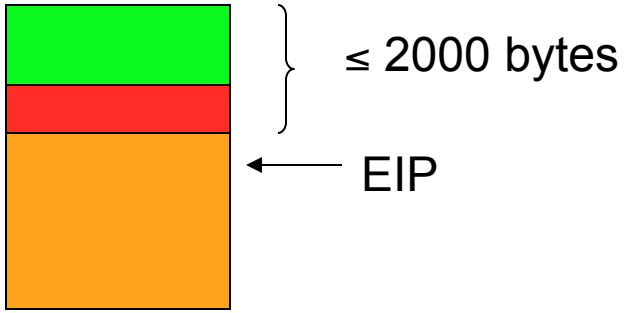
- To encourage diversity (i.e. different NOP and RET sizes)
- Raw Fitness / Niche Count.
  - Number
  - Distance



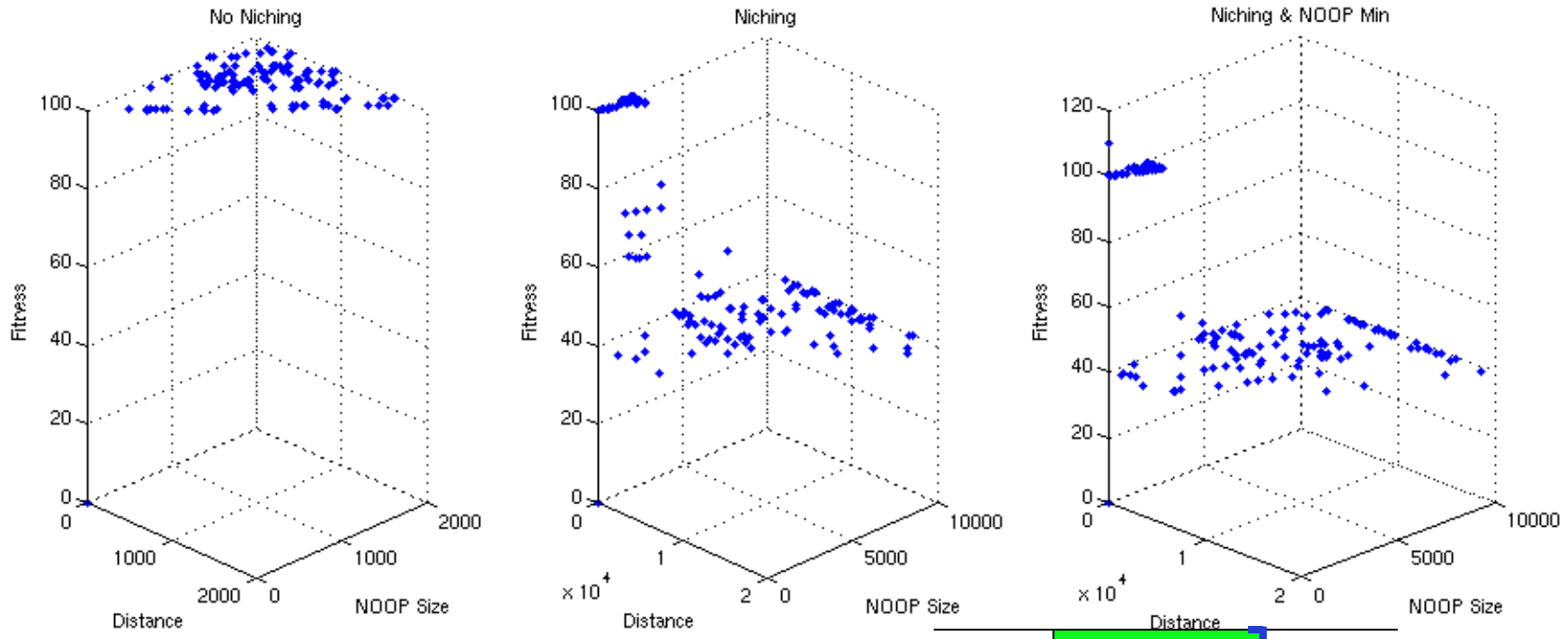
NOP vs. R.A. Length



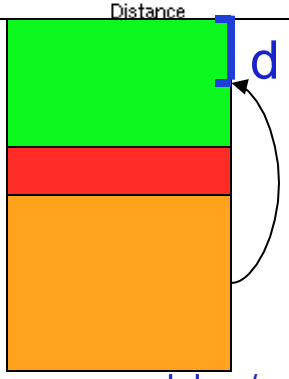
- Variable is 2000 bytes away from EIP



NOP vs. Accuracy

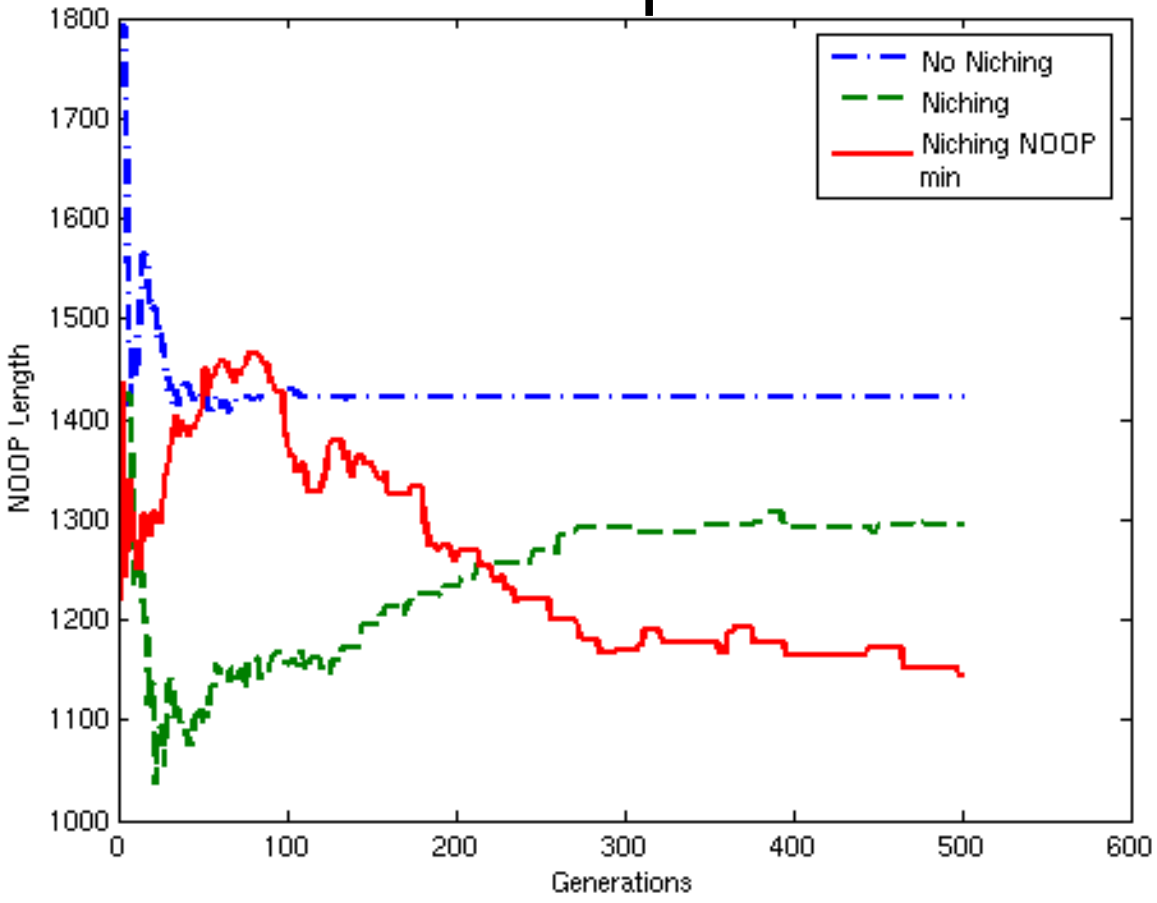


- As the return address gets more accurate, NOP sled gets smaller.



NOP Size

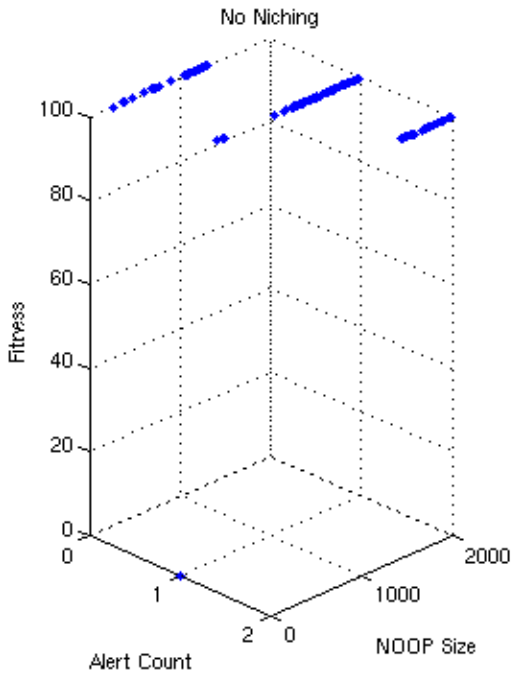
- NOP minimization improves.



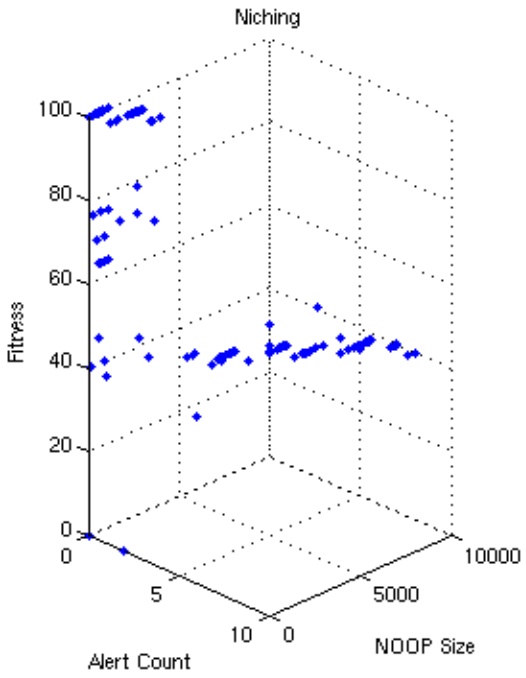
Snort IDS
www.snort.org

- Monitors network traffic.
- Signature based IDS.
- Widely used.
- All signatures disabled except 21 shellcode signatures.
- Emphasis on signature based detection.

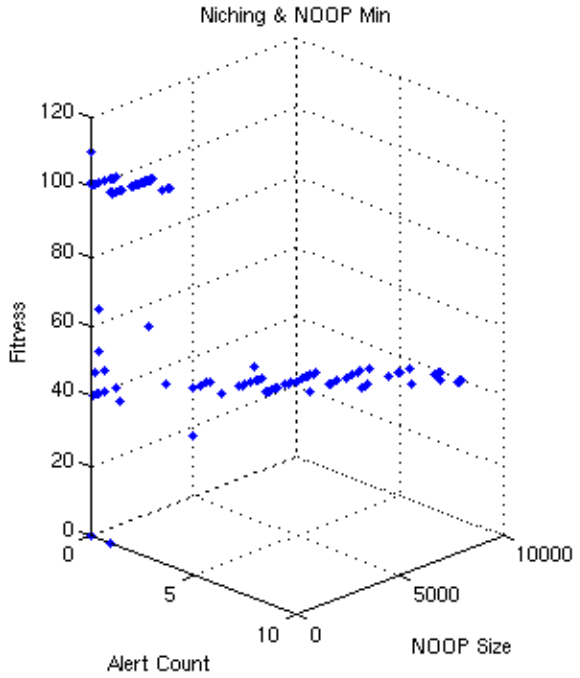
NOP vs. Alerts



FN=35%



FN=40%

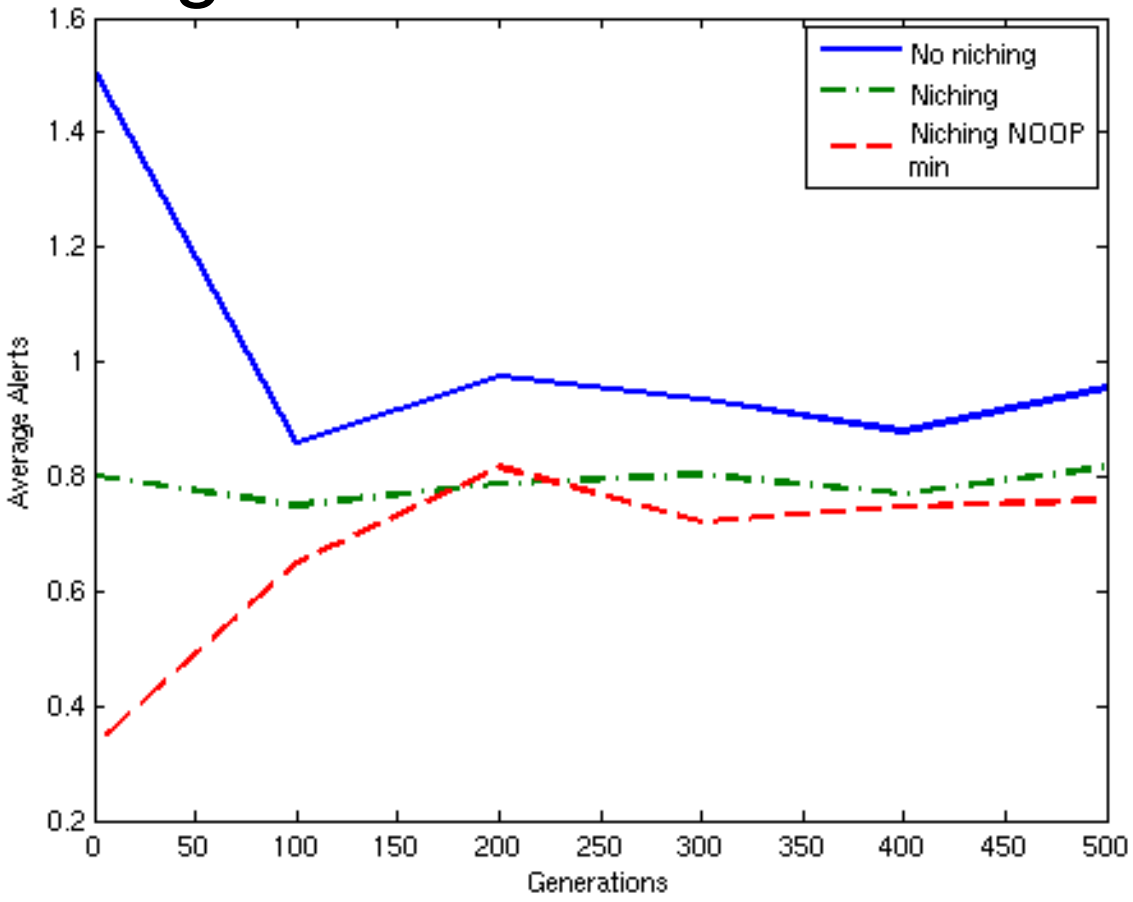


FN=52%

- Alerts raised per attack.
- Signature activated: NOP sled signature

Alerts

- Minimizing NOP reduces alerts.



Conclusion

- And… back to the arms race concept.
- Three experiments
  1. Basic GE (Mean fitness)
  2. GE with niching (More diverse than Basic GE)
  3. GE with niching and NOP minimization (Less detectable)
- Overwrite with an accurate address.
- NOP Length $\propto$ Accuracy

Questions?

- This research was conducted at Dalhousie Network Information Management and Security Laboratory.
- You can find more information about our research at: www.cs.dal.ca/projectx.

Snort Signature

```
alert ip $EXTERNAL_NET $SHELLCODE_PORTS ->
$HOME_NET any (msg:"SHELLCODE x86 NOOP";
content:"|90 90 90 90 90 90 90 90 90 90 90 90 90|";
depth:128; reference:arachnids,181;
classtype:shellcode-detect; sid:648; rev:7;)
```

Formulas

$$1 - \frac{\#\ of\ non\text{-}NoOP\ instructions}{NoOP\ sled\ length}$$

$$1 - \frac{\#\ of\ faulty\ return\ addresses}{Total\ \#\ of\ return\ addresses}$$

$$\frac{1}{\left| address_{actual} - address_{desired} \right| + 1}$$

$$\frac{1}{1 + NoOP\ sled\ length}$$

$$fitness = \mu_{shellcode} \times \begin{pmatrix} \mu_{success} \times (100 + W_{NS} \times score(NoOP)) + \\ (1 - \mu_{success}) \times \begin{pmatrix} W_{NE} \times score(NoOPError) + \\ W_{RE} \times score(retError) + \\ W_{DA} \times score(dist) \end{pmatrix} \end{pmatrix}$$

Formulas 2

$$f_{shared} = \frac{f_{raw}}{m_i}$$

$$m_i = \sum_{j=1}^{N} sh(d_{i,j})$$

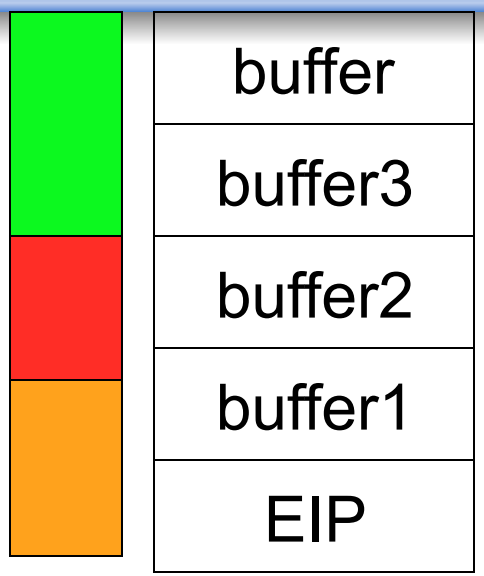
$$d_{i,j} = \sqrt{\left(NoOP_i - NoOP_j\right)^2 + \left(ret_i - ret_j\right)^2}$$

$$sh(d) = \begin{cases} 1 - \frac{d}{\sigma} & d < \sigma \\ 0 & otherwise \end{cases}$$

$$\sigma = \frac{\sqrt{\left(\max(NoOP) - \min(NoOP)\right)^2 + \left(\max(ret) - \min(ret)\right)^2}}{2\sqrt{q}}$$

Vulnerable Program

```
int main(int argc, char *argv[])
{
        char buffer1[500];
        char buffer2[500];
        char buffer3[500];
        char buffer[500];
        printf("Vulnerable : Variable at
          Addr : 0x%x\n", buffer);
        strcpy(buffer, argv[1]);
        return 0;
}
```



Grammar

```
code : exp
exp : detn detb deto alloc offsetc prel1 loop1 loop2 prel3 loop3 post3
digit : 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
number : digit + digit * 10 + digit * 100 + digit * 1000
detn : nsize = number ;
detb : bsize = nsize + number ;
deto : offset = number ;
alloc : buffer = malloc ( bsize );
offsetc: esp = sp();ret = esp - offset;
prel1 : ptr = buffer; addr_ptr = (long *) ptr;
loop1 : for ( i = 0 ; i < bsize ; i = i + 4 ) { exp1 };
loop2 : for ( i = 0 ; i < nsize ; i = i + 1 ) { exp2 };
prel3 : ptr = buffer + nsize;
loop3 : for ( i = 0 ; i < strlen (shellcode) ; i = i + 1 ) { exp3 };
post3 : buffer[ bsize - 1] = 0;
exp1 : *(addr_ptr++) = ret;
exp2 : buffer[ i ] = '\\x90';
exp3 : *(ptr++) = shellcode[ i ];
%%
```