StIns4CS: A State Inspection Tool for C#

Amjad Ibrahim

Sebastian Banescu

Affiliation

Prof. Dr. Alexander Pretschner

Technische Universität München

Fakultät für Informatik

Lehrstuhl XXII für Software Engineering

INTRODUCTION

Software Tampering

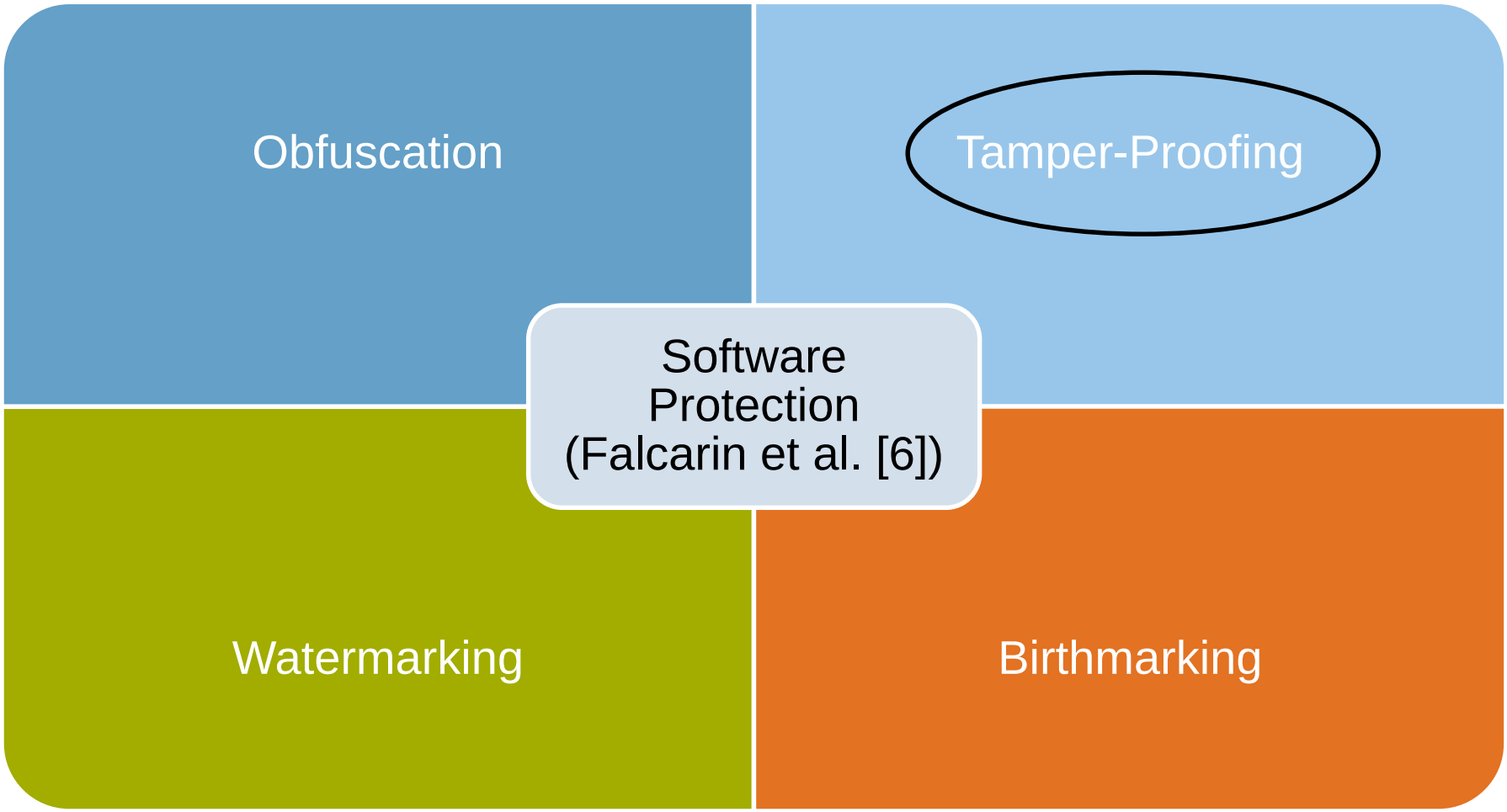
Context: Running an application on a platform controlled by an adversary

Attacks: Unauthorized modification of software

Consequence: Retail value of pirated software is $63 billion in 2013 [13]

```
1 public float sum(float [] arr){
2  float sum = 0;
3  for (int i = 0; i < arr.Length; i++){
4        // subtracts 0.00005
5        arr[i] = round(arr[i]);
6        sum += arr[i];
7 }
8 return sum;
9 }
```

How to protect a software



Contributions

- The design and implementation of StateInspection4CSharp tool.

- Symbolic execution to generate the set of input-output pairs for a certain set of functions.
- Evaluation of the performance and stealth of StIns4CS with various response mechanisms.
- A method to increase the stealth of the response mechanism by degrading results.

StIns4CS

The Concept

- State Inspection [4]
- Integrity of pure functions
- Random networks of guards
- A configurable response mechanism

Guard

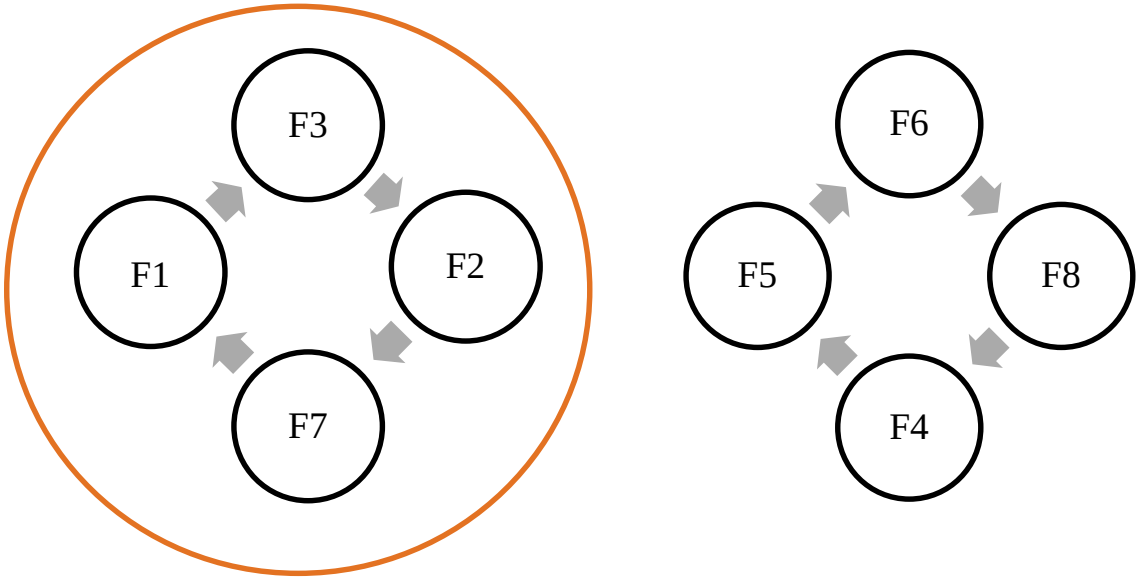
```
1 // Create an instance of the target class
2   A instance = new A();
3   var output;
4 // Invoke the checked function "f1"
5   output = instance.f1(param1);
6 // Compare against known value
7   if (output != expectedValue){
8         callResponse();
9   }
```

Code Guard Network

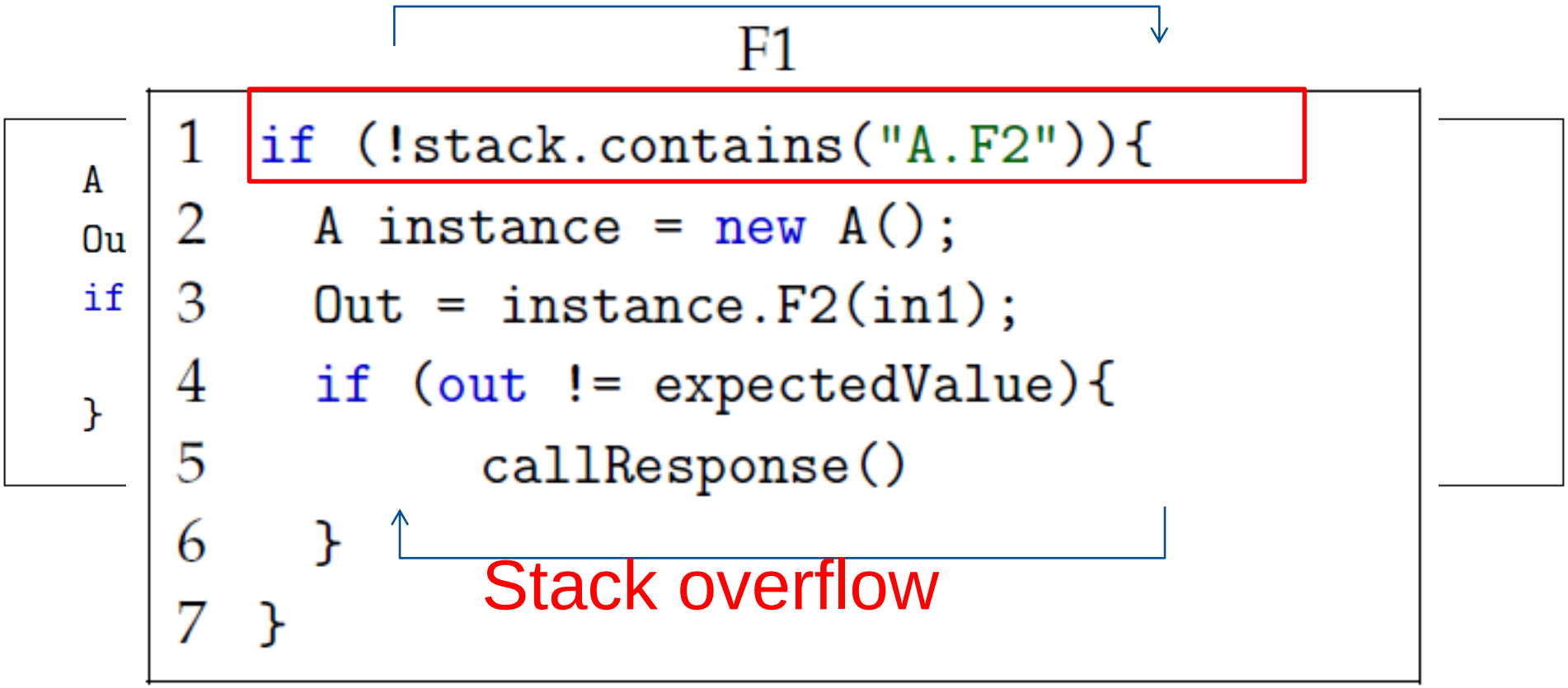
Original Methods

[F1, F2, F3, F4, F5, F6, F7, F8] → [F3, F2, F7, F1, F6, F8, F4, F5]

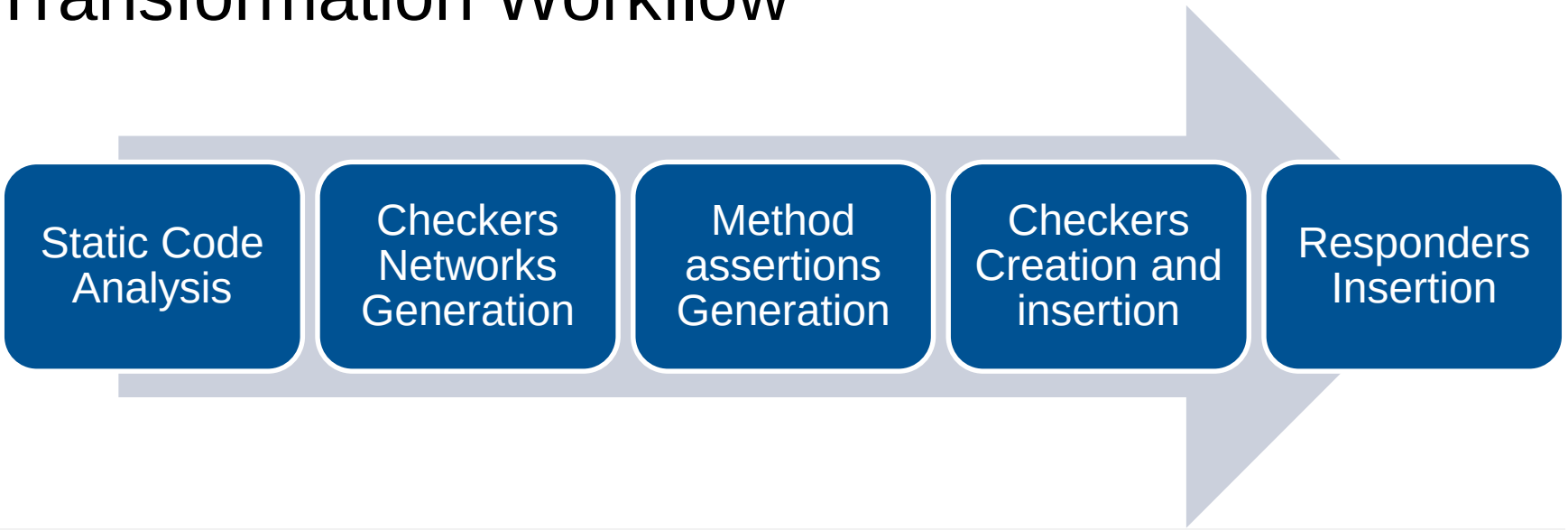
Shuffled Methods



Stack Inspection



Transformation Workflow



- Responses in case of detecting a tampering
  - Immediate crash
  - Delayed crash
  - Do nothing
  - Remote logging

Primitive combination

- Sabotaging the return values
- Incorporating results of the check with return value

Benefits

1. Hides the comparison
2. Hides the response
3. Adds diversity to the protection code

Primitive combination- Example

```
1 // Create an instance of the target class
2  A instance = new A();
3  var output;
4 // Invoke the checked function "f1"
5  output = instance.f1(param1);
```

```
return result.Substring(expected-output);
```

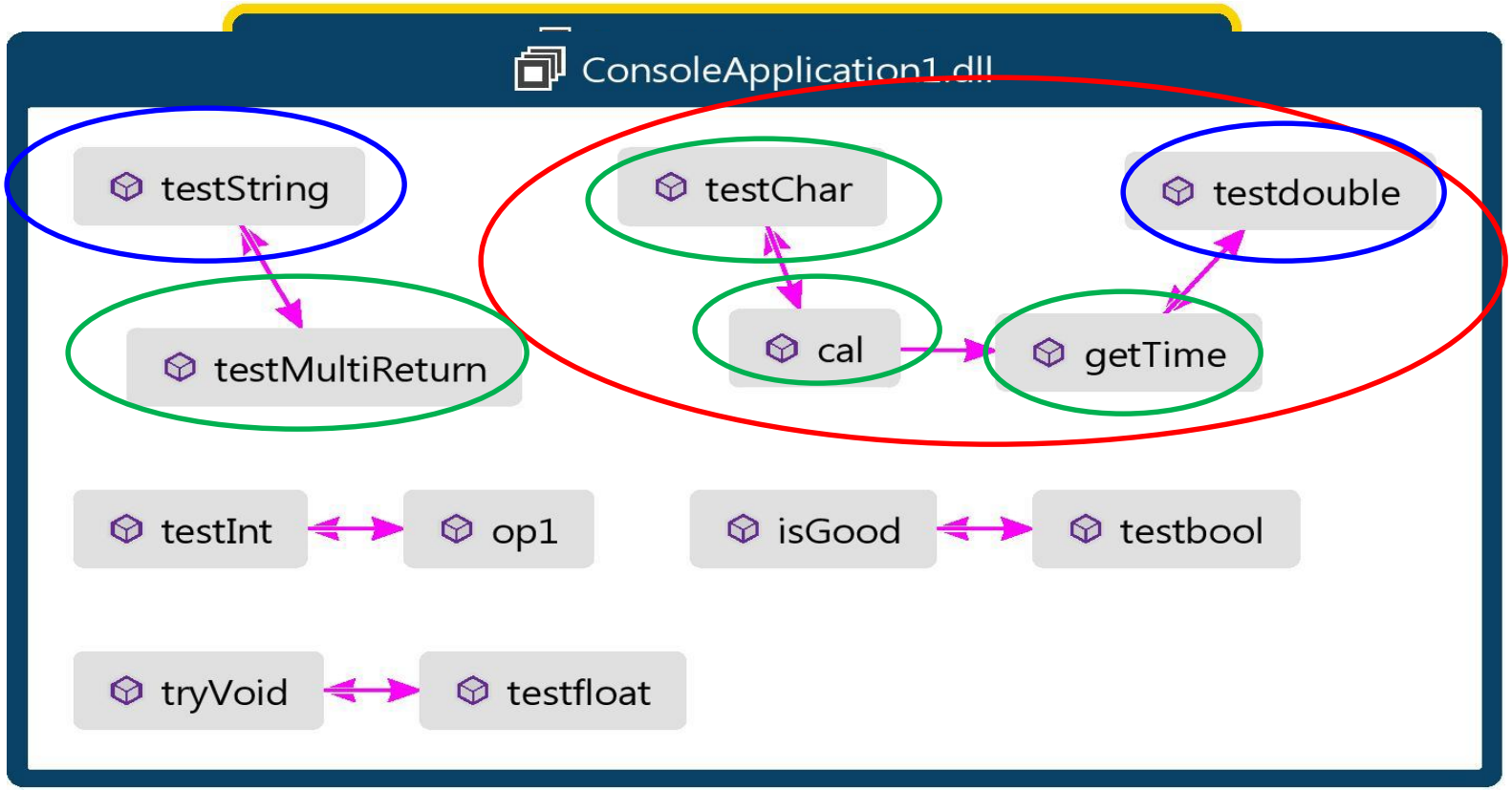
EVALUATION

- Effectiveness against static and dynamic patching
- Stealth of checking code
- Cost of checking

Effectiveness against code patching

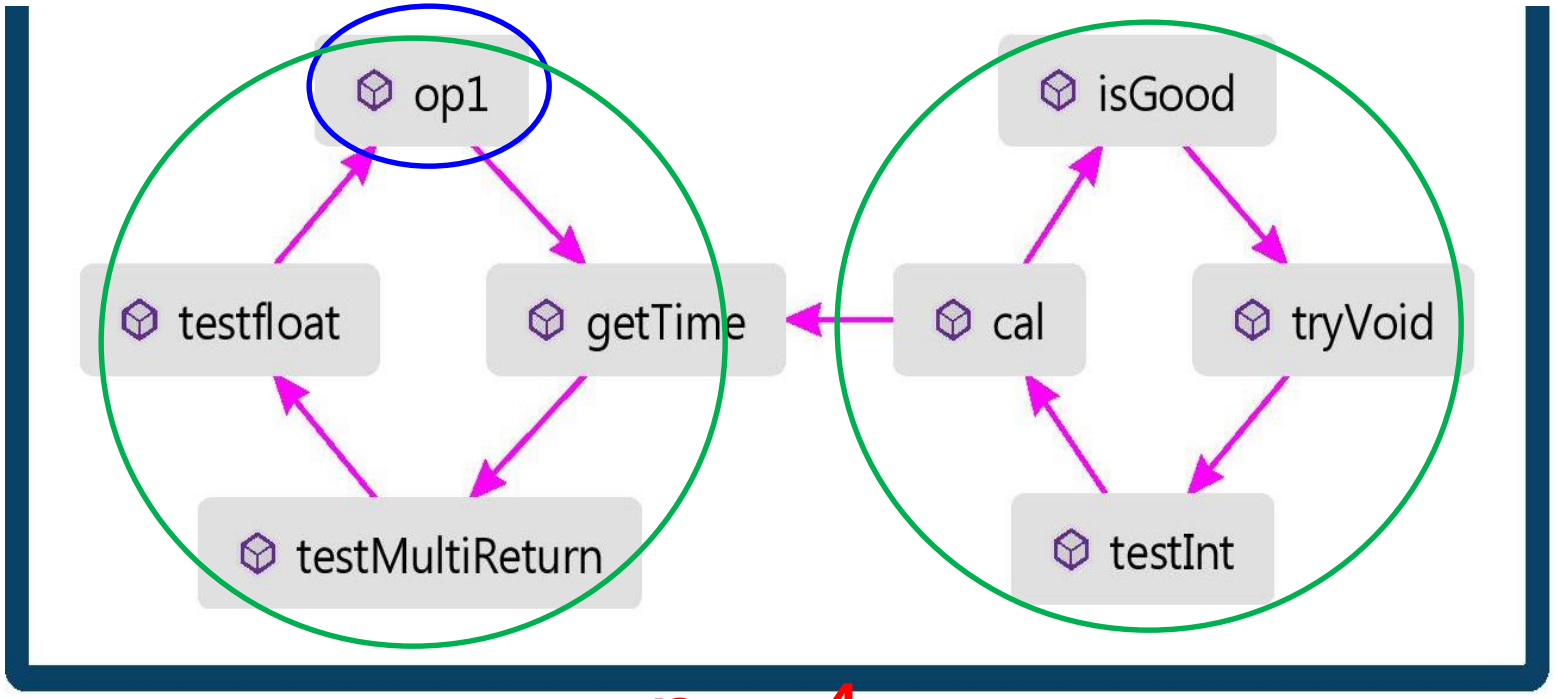
- Verify how the applications respond to patching
  - Static code patching
  - Dynamic memory patching
- Test variable:
  - Component size

Discussion



n = 2

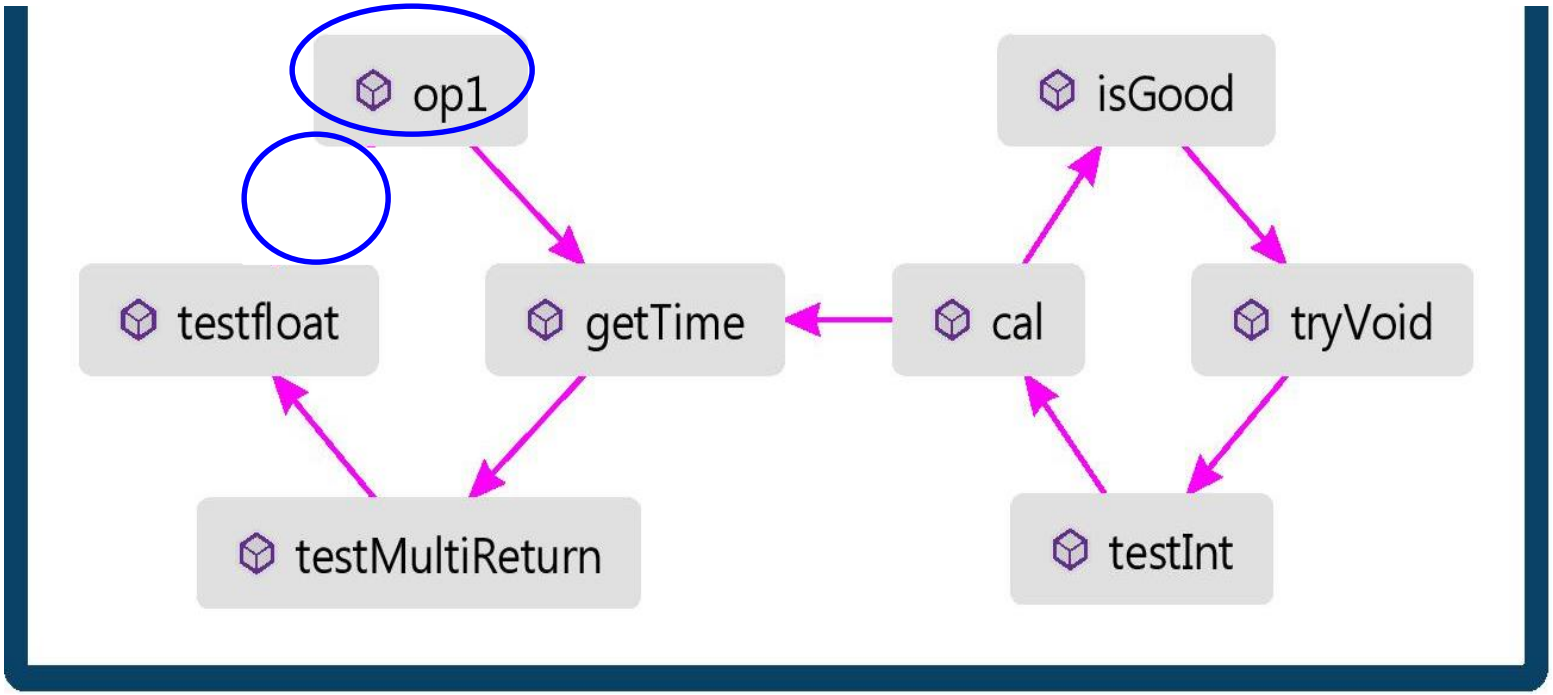
Discussion



n = 4

Patching detected in at least n-1 places

Disabling a check



n = 4

Stealth of checking code

- Pattern matching attacks
- Test variables
  - Primitive combination
  - Virtualization obfuscation
- Three regular expressions:
  - Regex1: any if-statement
  - Regex2: if-statement with stack inspection
  - Regex3: response call statement

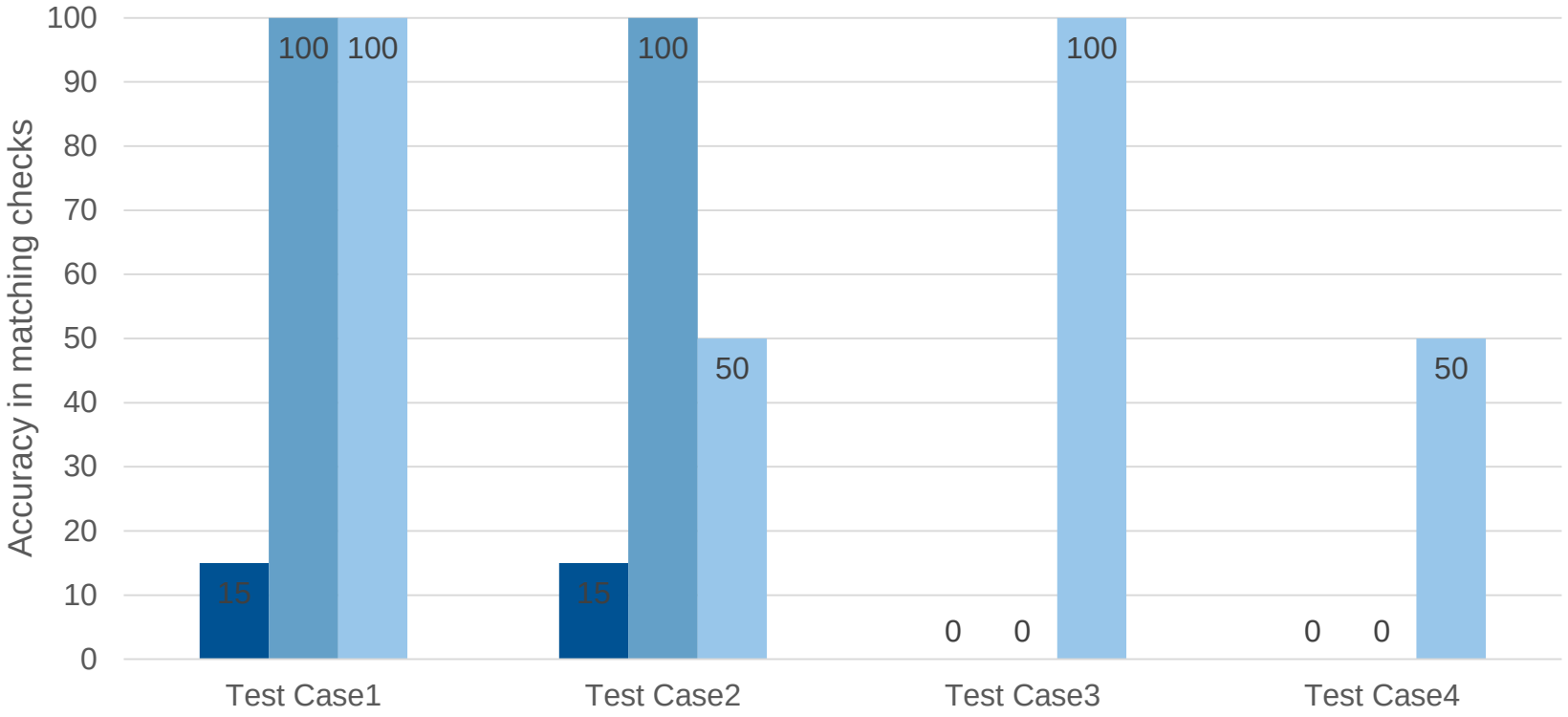
Discussion

Regex1 Regex2 Regex3

F1

```
1  if (!stack.contains("A.F2")){
2     A instance = new A();
3     Out = instance.F2(in1);
4     if (out != expectedValue){
5           callResponse()
6     }
7  }
```

Results

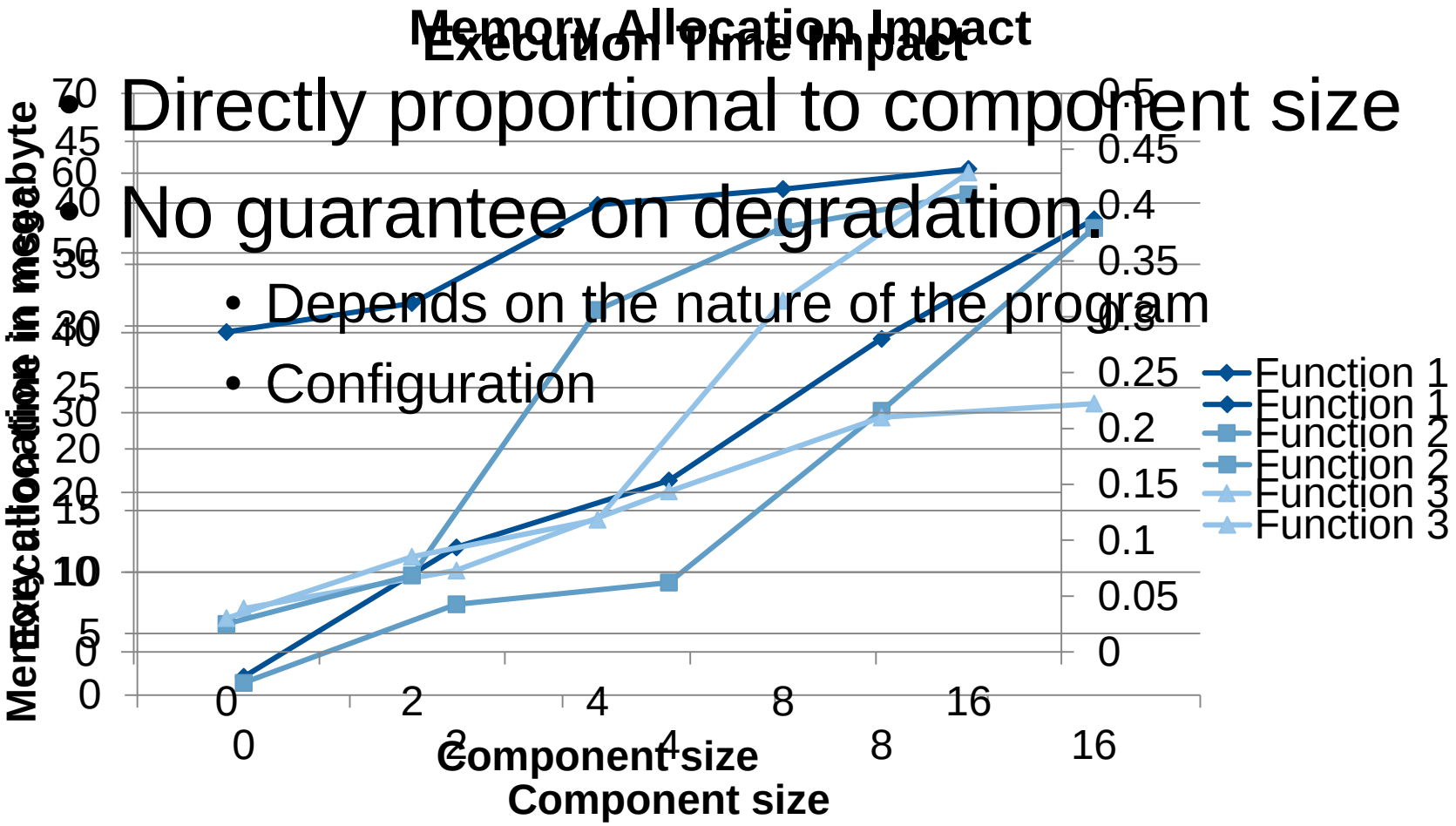


Name	Primitive_Combination	Virtualization
Test case 1	No	No
Test case 2	YES	No
Test case 3	No	YES
Test case 4	YES	YES

Cost of checking

- Function invocations added
  → performance will be affected
  - Execution time
  - Memory allocation
  - Code size

Execution time and memory allocation



Directly proportional to component size

No guarantee on degradation.

- Depends on the nature of the program
- Configuration

CONCLUSION & FUTURE WORK

Conclusion

- Effectiveness of the approach
  - StIns4CS
- Solving problems affect stealth
  - Stack inspection
- Enhancing the stealth is possible
  - Primitive combination

Future work

- Stack inspection → other options to break cycles
  - Information from the code call graph
- Primitive combination is conditioned
  - Generalizing this concept
- Software-diversity techniques within the checks
  - Versions of the same test

Thanks for your attention

ibrahim@in.tum.de

QUESTIONS!

References

[1] Chang, Hoi, and Mikhail J. Atallah. “Protecting Software Code by Guards.” In Security and Privacy in Digital Rights Management, 160–175. Springer, 2002.

[2]. Horne, Bill, Lesley Matheson, Casey Sheehan, and Robert E. Tarjan. “Dynamic Self-checking Techniques for Improved Tamper Resistance.” In Security and Privacy, 141–159. Springer, 2002.

[3]. Giffin, Jonathon T., Mihai Christodorescu, and Louis Kruger. “Strengthening Software Self-checksumming via Self-modifying Code.” In Computer Security Applications Conference, 21st Annual, 10–pp, 2005.

[4]. Mavrogiannopoulos, Nikos, Nessim Kisserli, and Bart Preneel. “A Taxonomy of Self-modifying Code for obfuscation.” Computers & Security 30, 2011.

[5]. David Aucsmith , “Tamper resistant software: an implementation”, in information hiding, 199

[6] P. Falcarin, C. Collberg, M. Atallah, and M. Jakubowski. Guest editors’ introduction: Software protection. Software, IEEE, 28(2):24–27, 2011.

[7]. L. Martignoni, R. Paleari, and D. Bruschi. Conqueror: tamper-proof code execution on legacy systems. In Proceedings of the 7th Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), Lecture Notes in Computer Science. Springer, July 2010.

References

[8]. J. Qiu, B. Yadegari, B. Johannesmeyer, S. Debray, X. Su, “Identifying and Understanding Self-Checksumming Defenses in Software”, 2015.

[9]. A. Seshadri , M. Luk , E. Shi , A. Perrig , L. van Doorn , P. Khosla, “Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems”, Proceedings of the twentieth ACM symposium on Operating systems principles, 2005.

[10]. G. Tan, Y. Chen, M.H. Jakubowski, ”Delayed and controlled failures in tamper-resistant systems”, 8th Information Hiding, Lecture Notes in Computer Science, LNCS, vol. 4437 (2006).

[11]. G. Wurster , P. C. van Oorschot , A. Somayaji, “A Generic Attack on Checksumming-Based Software Tamper Resistance”, 2005 IEEE Symposium on Security and Privacy.

[12] Collberg, Christian,“Surreptitious software obfuscation, watermarking, and tamperproofing for software protection, 2010.

[13] S. Smith and S. Weingart. Building a high performache programmable secure coprocessor.Computer Networks,1999.

[14] Steve R. White and Liam Comerford. ABYSS: An architecture for software protection. IEEE Transactions on Software Engineering,1990.

[15] Bennett Yee and J. D. Tygar. Secure coprocessors in electronic commerce applications. 1995.

[16]. http://globalstudy.bsa.org/2011/downloads/study_pdf/2011_BSA_Piracy_Study-InBrief.pdf

[17]. http://research.microsoft.com/en-us/projects/pex/

[18]. http://roslyn.codeplex.com

[19]. https://dzone.com/articles/smart-continuous-delivery