Source Code Analysis and Application Security

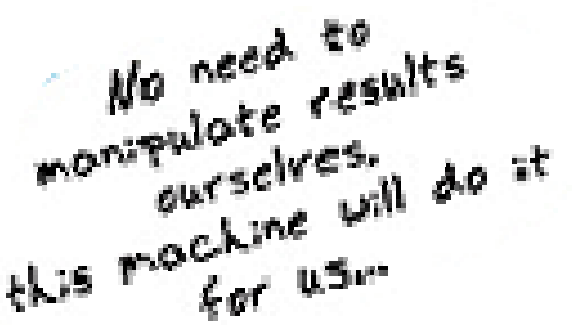
Maty SIMAN, CISSP
Founder, CTO
Checkmarx

maty@checkmarx.com

Checkmarx
Security from the source

Agenda

- Source Code Analysis
- SCA and Security
- Demo



http://oeronline.com/kk-cartoon/images/april_2006bg.jpg

Source Code Analysis

“Static code analysis is the analysis of computer software that is performed without actually executing programs…”

Wikipedia

Source code analysis

Two types

- Simple – code as text
- Complex – i ation

```
public class MessageFilter:System.Windows.Forms.IMessageFilter
{
	...
	public bool PreFilterMessage(ref System.Windows.Forms.Message m)
	{
		Debug.Assert(frmMain != null);
		Debug.Assert(frmCtrlContainer != null);

		Control ctrl= Control)Control.FromHandle(m.HWnd);
		if(frmCtrlContainer.Contains(ctrl) && m.Msg == WM_PAINT)
		{
			//let the main form redraw other sub forms, controls
			frmMain.Refresh();
			//prevent the controls from being painted
			return true;
		}
		return false;
	}
}
//
public class SelectionUIOverlay : System.Windows.Forms.Control
{
	...
	private void InvalidateEx()
	{
		Invalidate();
		//let parent redraw the background
		if(Parent==null)
			return;
		Rectangle rc=new Rectangle(this.Location,this.Size);
		Parent.Invalidate(rc,true);
		...
		//move and refresh the controls here
	}
}
```

Source Code Analysis

- Text
  - Finding using RegEx
  - Some metrics

*"**When you can measure** what you are speaking about, and express it into numbers, **you know something** about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science."*

Lord Kelvin (1824-1904)

Source Code Analysis

- Text
  - Finding using RegEx
  - Some metrics
  - Coding standards
    - Microsoft style:
      if (2 == a) ….

Intermediate representation

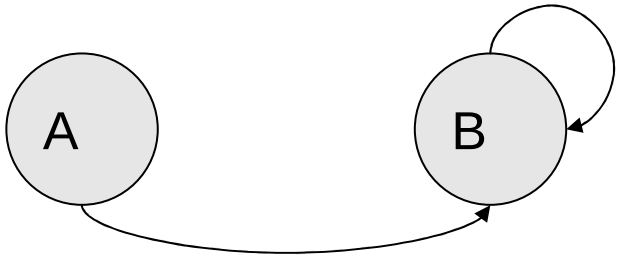
- Call Graph
- CFG
- CDG
- DFG
- Slicing

Call graph

- Directed graph
- Each vertex represents either calling or called element (class, method, page)
- Vertex A is connected to vertex B using the directed edge E iff A calls B

```
void A ()
{
    B();
}
```

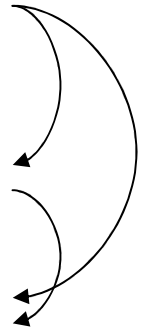
```
void B ()
{
    B();
}
```



Control Flow Graph

- Each vertex in the graph represents a basic block – a straight line of code without any jumps or jumps targets (it is guaranteed that all commands within a block are executed in their order).
- Directed edges are used to represent jumps in the control flow.

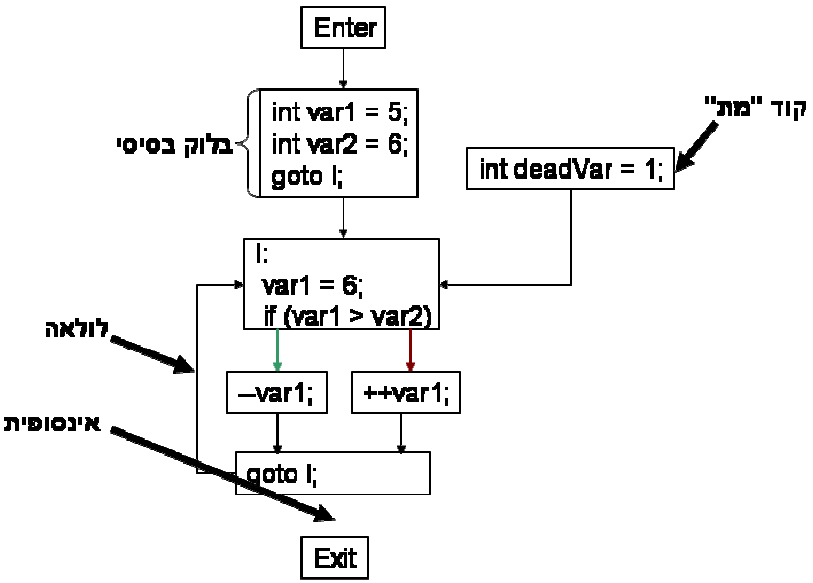
```
a = 3;
b = 5;
c = a;
If (b > a)
        c = b;
printf (“%d”, a)
printf (“%d”, b)
printf (“Max: %d”, c)
```



Intermediate representation

- CFG

```
void main()
{
    int var1 = 5;
    int var2 = 6;
    goto label;
    int deadVar1 = 1;
label:
    var1 = 6;
    if (var1 > var2)
    {
        --var1;
    }
    else
    {
        ++var1;
    }
    goto label;
}
```



Control Dependence Graph

- Enhances CFG.
- Each LOC has its own vertex
- Edge B is directed by edge A iff the execution if B depends on the execution of A

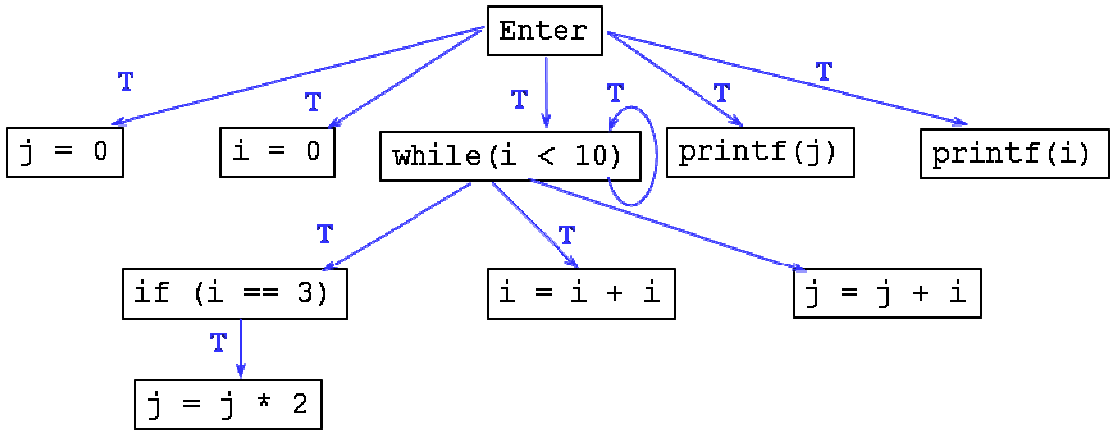
Example

Control Dependence Graph (cont.)

```
void main()
 {
    int j = 0;
    int i = 0;

    while (i < 10){
        if (i == 3){
            j=j*2;
        }
        j = j + i;
        i = i + 1;
    }

    printf ("%d\n", j);
    printf ("%d,n", i);
}
```



Data Flow Graph

- Represents the flow of data through code.
- Each LOC has its own vertex (like CDG).
- Edge represents **direct** influence of data in the source vertex on the data in the destination vertex (therefore, assignment statements are source vertexes)

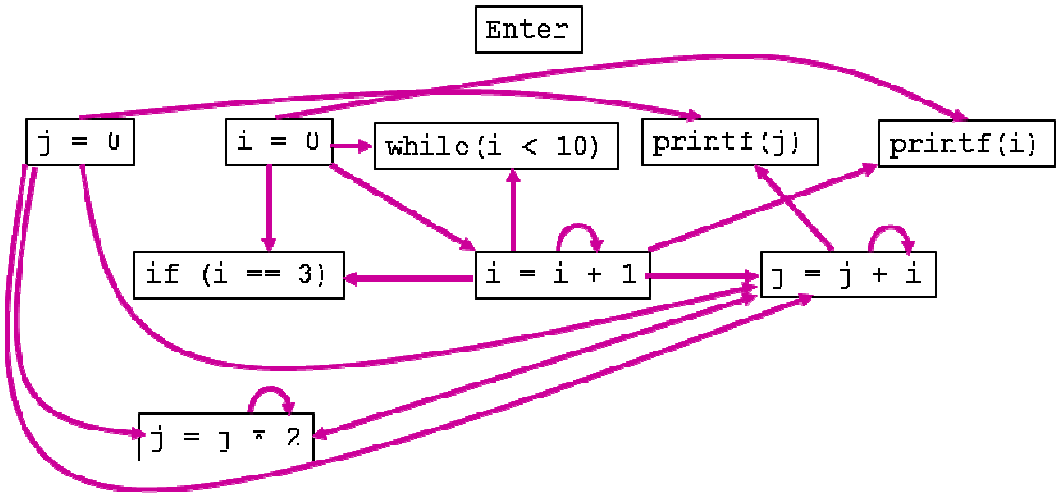
Example

Control Dependence Graph (cont.)

```
void main()
 {
    int j = 0;
    int i = 0;

    while (i < 10){
        if (i == 3){
            j=j*2;
        }
        j = j + i;
        i = i + 1;
    }

    printf ("%d\n", j);
    printf ("%d,n", i);
}
```



Slicing

- Finding a relevant subset of the application

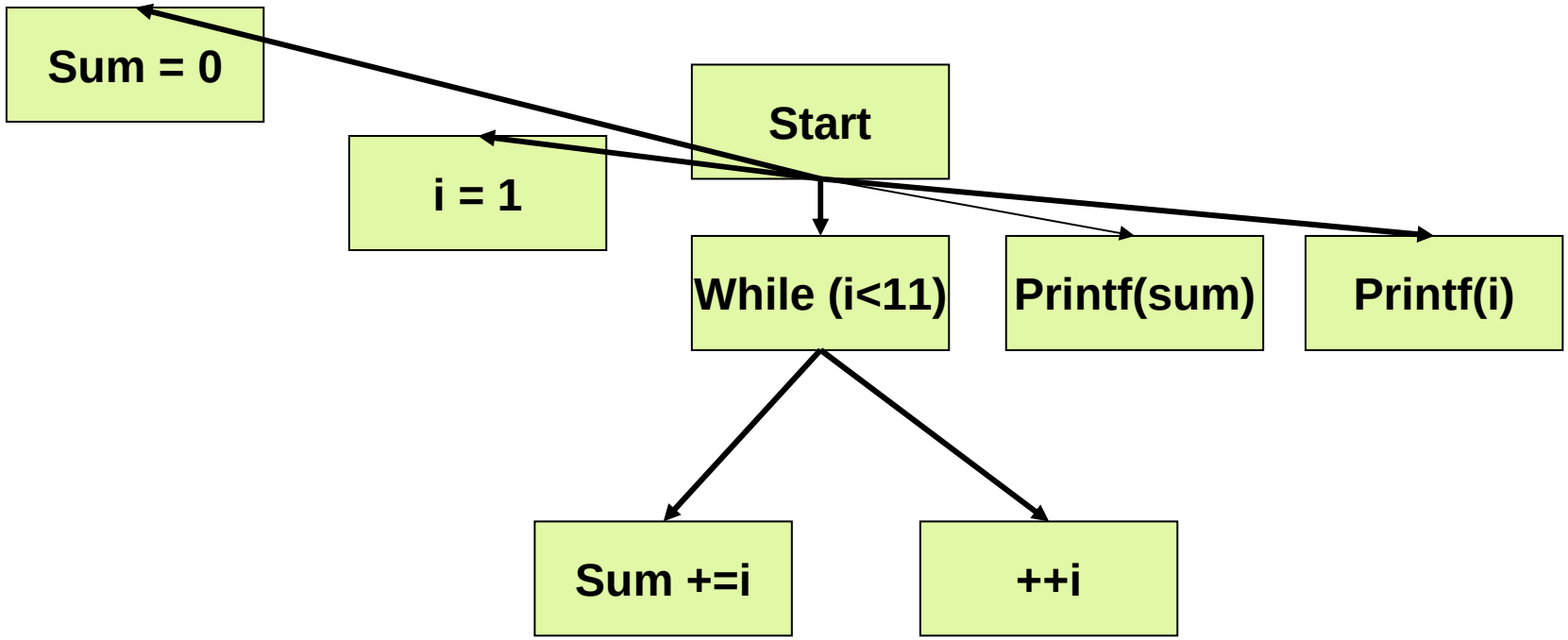
```
void main()
{
   int sum = 0;
   int i = 1;
   while (i < 11)
   {
       sum = sum + i;
       i = i + 1;
   }
   printf (“%d\n”, sum);
   printf (“%d\n”, i);
}
```

Slicing (cont.)

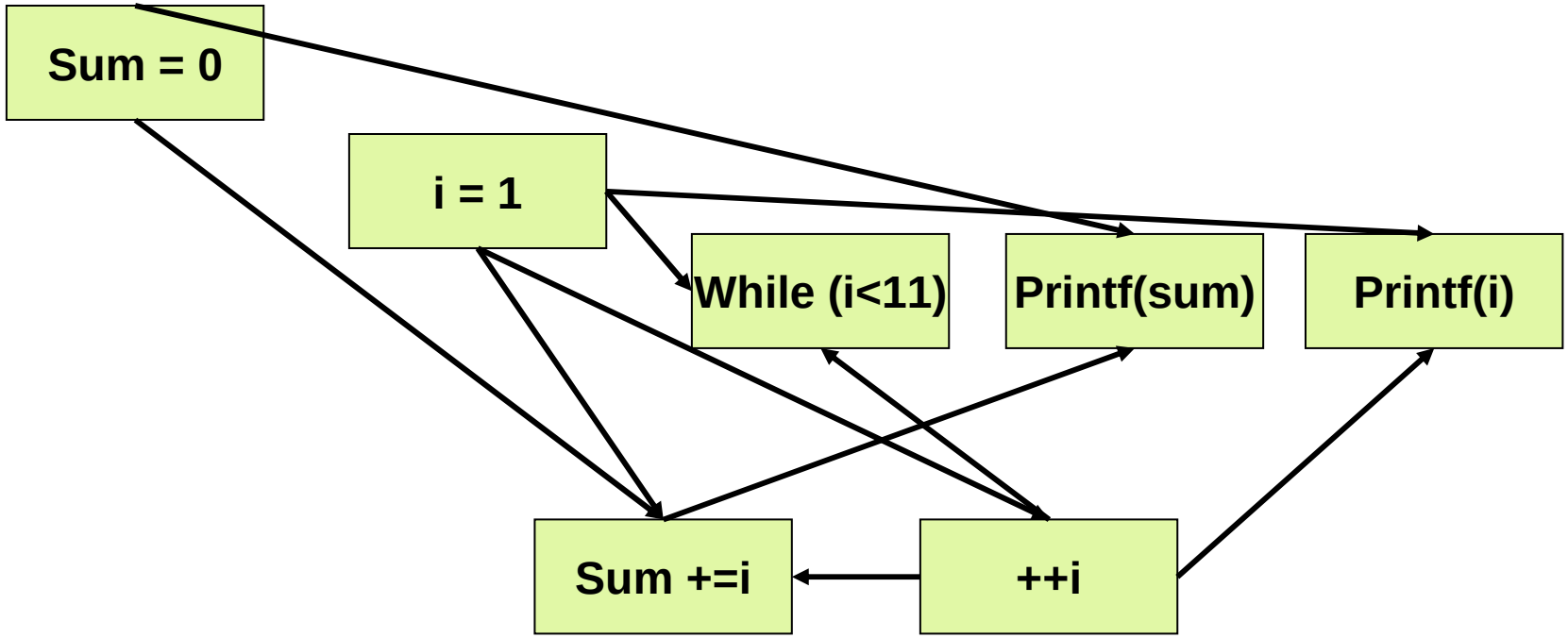
- Finding a relevant subset of the application

```
void main()
{
   int sum = 0;
   int i = 1;
   while (i < 11)
   {
       sum = sum + i;
       i = i + 1;
   }
   printf (“%d\n”, sum);
   printf (“%d\n”, i);
}
```

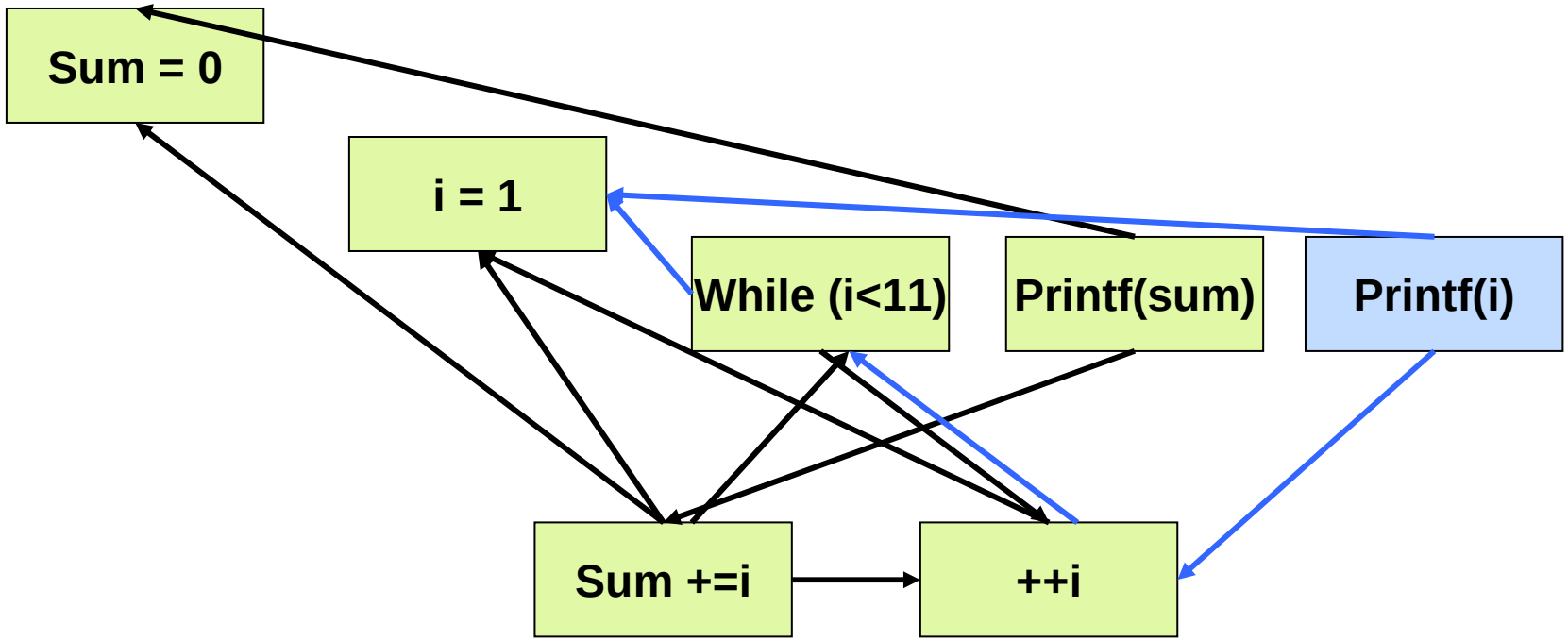
CDG



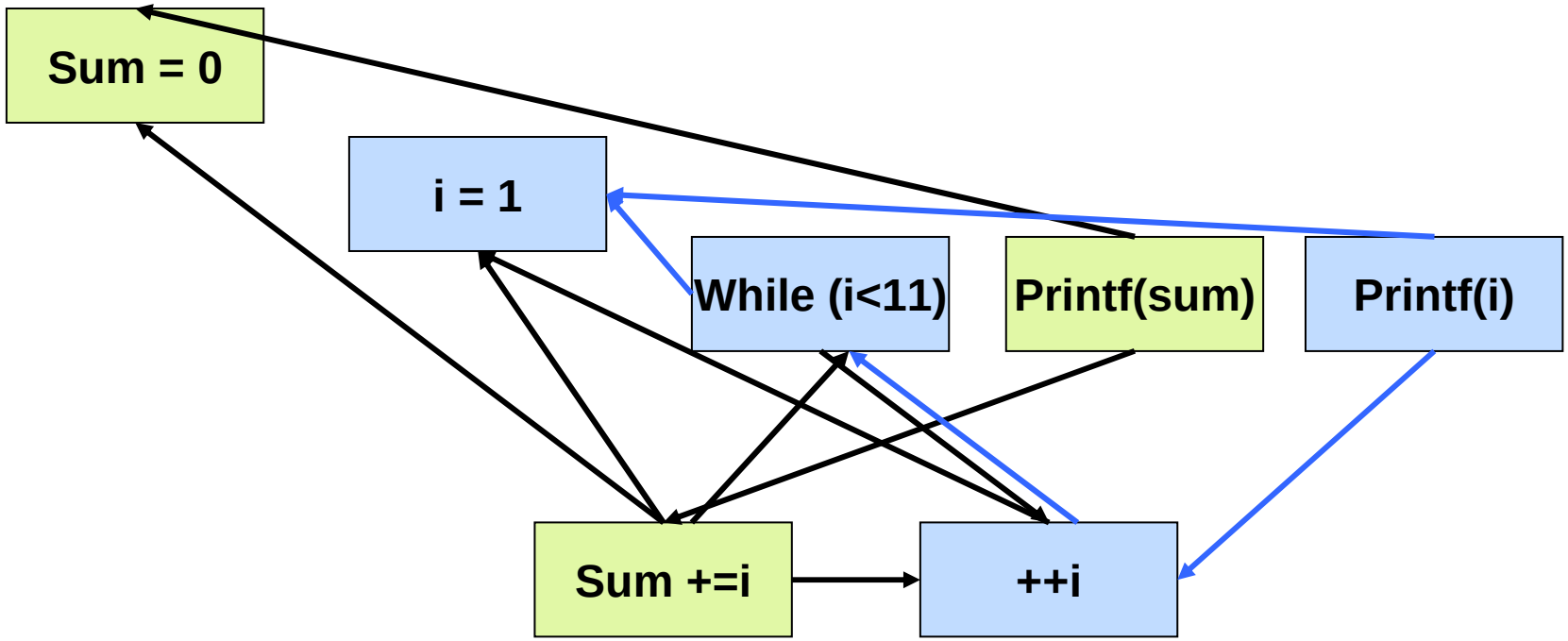
DFG



(DFG+CDG)’



(DFG+CDG)’



Virtual machine

- Static ?!

"Well well well... If it isn't Pamela Anderson!"

http://www.cartoonstock.com/lowres/iba0013l.jpg

Limitations

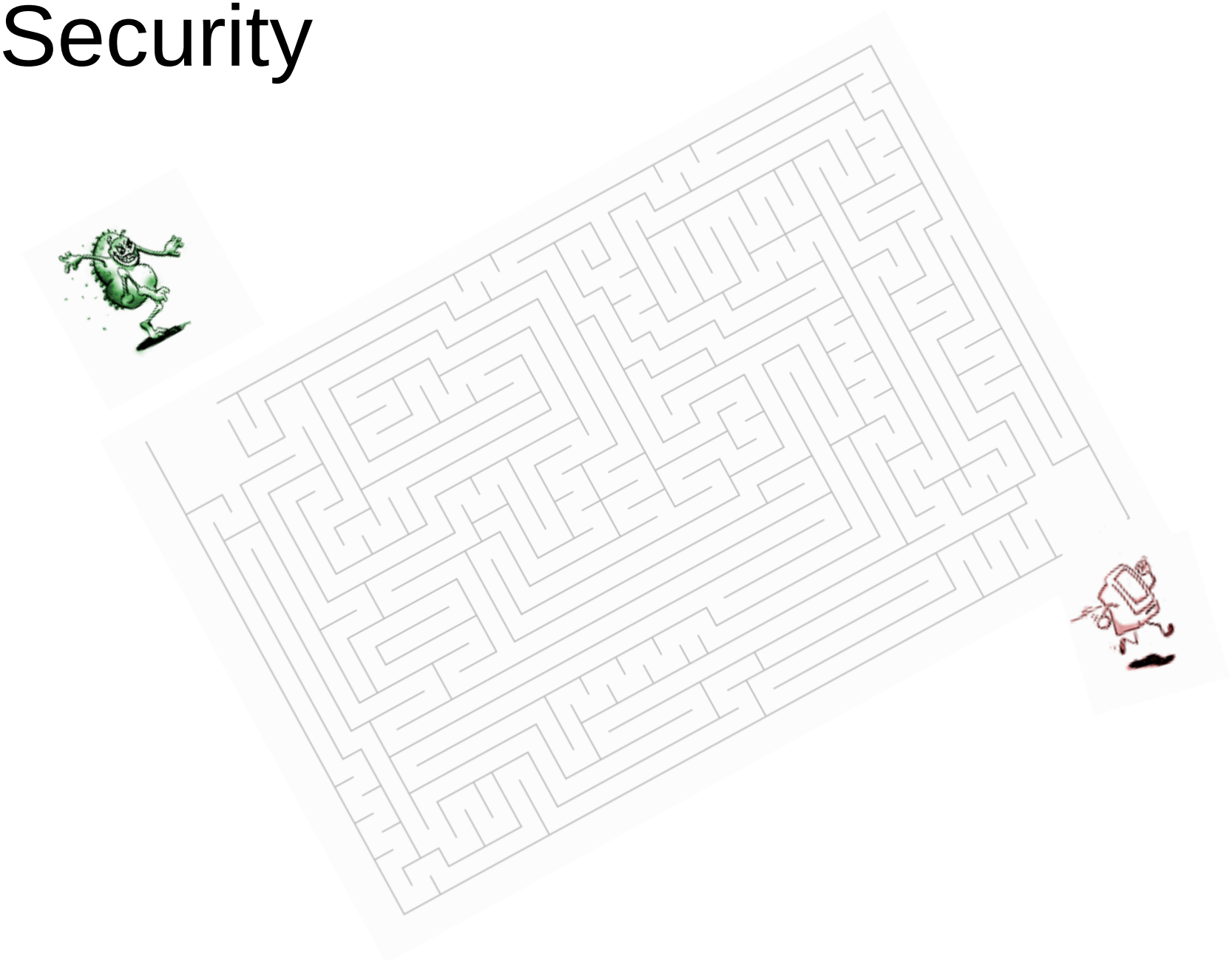
More

- SAT Solvers
- Parallel
- Optimization
- Migrations
- .
- ..
- …

Related techniques

- Code review
- Dynamic analysis
- Testing
- Debugging

Security



Security
pros/cons

- Penetration testing
- Code review
- WAF

WHITEHAT™ SECURITY

AUTOMATED SCANNING VS. THE OWASP TOP TEN

The OWASP Top Ten[1] is a list of the most critical web application security flaws – a list also often used as a minimum standard for web application vulnerability assessment (VA) and compliance. There is an ongoing industry dialog about the possibility of identifying the OWASP Top Ten in a purely automated fashion (scanning). People frequently ask what can and can't be found using either white box or black box scanners. This is important because a single missed vulnerability, or more accurately exploited vulnerability, can cause an organization significant financial harm. Proper expectations must be set when it comes to the various vulnerability assessment solutions.

For our part, WhiteHat Security is in the website security business and provides a vulnerability management service. Our Sentinel Service incorporates expert analysis and scanning technology. Using a black box process, we assess hundreds of websites a month, more than anyone in the industry. What we've come to understand is a significant portion of vulnerabilities are virtually impossible[2] for scanners to find. By the same token even the most seasoned web security experts cannot find many issues in a reliable and consistent manner. To achieve full vulnerability coverage we must rely on a combination of both methods.

WHITEHAT™ SECURITY

AUTOMATED SCANNING VS. THE OWASP TOP TEN

A1 Unvalidated Input

Scanners must hazard a guess about what "transfer.cgi" does. Otherwise it would be impossible to determine what it should NOT do.

Humans can easily figure this out, but scanners aren't that intelligent. There is no knowledge of context. For the sake of discussion, let's say a scanner has the ability, because there's a dollar figure present and "transfer" keyword in the URL might help it decide that this feature moves money. Realistically, these parameter names could be anything and often far more cryptic. To attempt a classic funds transfer attack, let's change the above URL substituting the "1000.00" amount to "-1000.00."

Negative Amount Example:

http://server/transfer.cgi?
from_acct=1235813&to_acct=31415&amount=-1000.00&session=1001

By transferring a negative amount, the web application would potentially deduct money from the target account instead of adding to it! The challenge for a scanner is being able to decide whether or not the attack succeeded. How can it tell?

If the fraudulent transfer succeeded the website might respond with, "Success, would you like to make another transaction?," "Transfer will take place by 9 am tomorrow," "Request received, thank you," or any number of possible affirmations. If the attack failed, "Transfer failed", "Error: Transfer amount must be a positive number", or "Bank robbery detected, men with guns have been dispatched to your location!" Every custom Web Bank application will likely respond in a different manner. That's the problem! Pre-programming in all the possible keyword phrases or behavioral aspects is simply unfeasible and for all mathematical provability, impossible. However, human gray matter can make this determination.

Man vs. Machine

“Brain” intensive

virtually impossible for scanners to find.

Time intensive

experts cannot find many issues in a reliable and consistent manner.