

Who are we?

- Co-founder and Chief Scientist at Lastline, Inc.
 - Lastline offers protection against zero-day threats and advanced malware
- Professor in Computer Science at UC Santa Barbara
 - many systems security papers in academic conferences
- Member of Shellphish



Who are we?

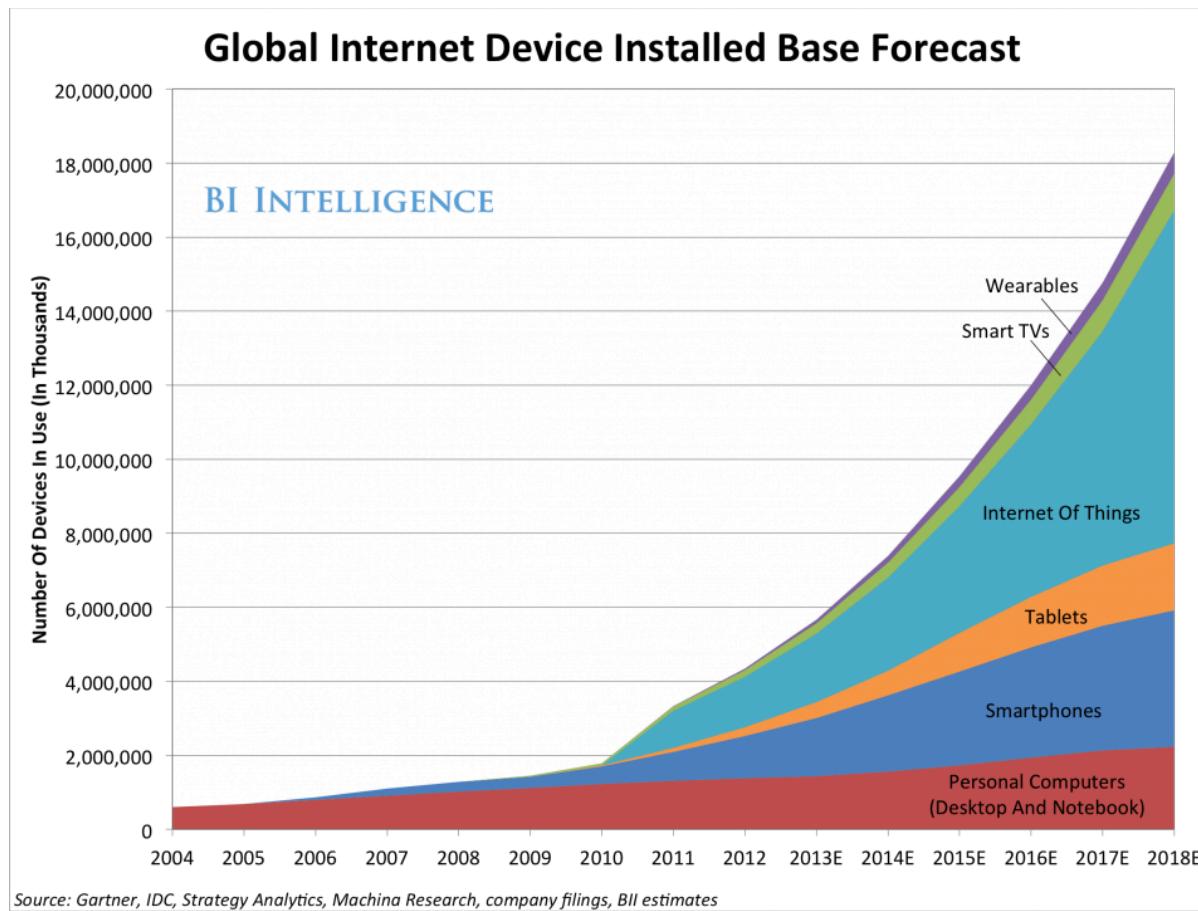
- PhD Student at UC Santa Barbara
 - research focused primarily on binary security and embedded devices
- Member of Shellphish
 - team leader of Shellphish's CGC effort



A horizontal banner at the top of the slide features a dark blue background. On the left side, there is a faint, stylized illustration of a circuit board with various lines and nodes. On the right side, there are several bright, white, jagged lines resembling lightning bolts or plasma arcs, which are more concentrated towards the center-right of the banner.

What are we talking about?

The “Internet of Things”



Embedded software is everywhere



What is on embedded devices?

- Embedded Linux and user-space programs
- Custom OS and custom programs combined together in a *binary blob*
 - typically, the binary is all that you get
 - and, sometimes, it is not easy to get this off the device

Binary Analysis

Binary analysis 

noun | bi·na·ry anal·y·sis | \'bī-nə-rē ə-'na-lə-səs\

1. The process of automatically deriving properties about the behavior of binary programs
2. Including static binary analysis and dynamic binary analysis

Goals of Binary Analysis

- Program verification
 - Program testing
 - Vulnerability excavation
 - Vulnerability signature generation
-
- Reverse engineering
 - Vulnerability excavation
 - Exploit generation



Static Binary Analysis

- reason over multiple (all) execution paths
- can achieve excellent coverage
- precision versus scalability trade-off
 - very precise analysis can be slow and not scalable
 - too much approximation leads to wrong results (false positives)
- often works on abstract program model
 - for example, binary code is lifted to an intermediate representation

Dynamic Binary Analysis

- examine individual program paths
- very precise
- coverage is (very) limited
- sometimes hard to properly run program
 - hard to attach debugger to embedded system
 - when code is extracted and emulated, what happens with calls to peripherals?



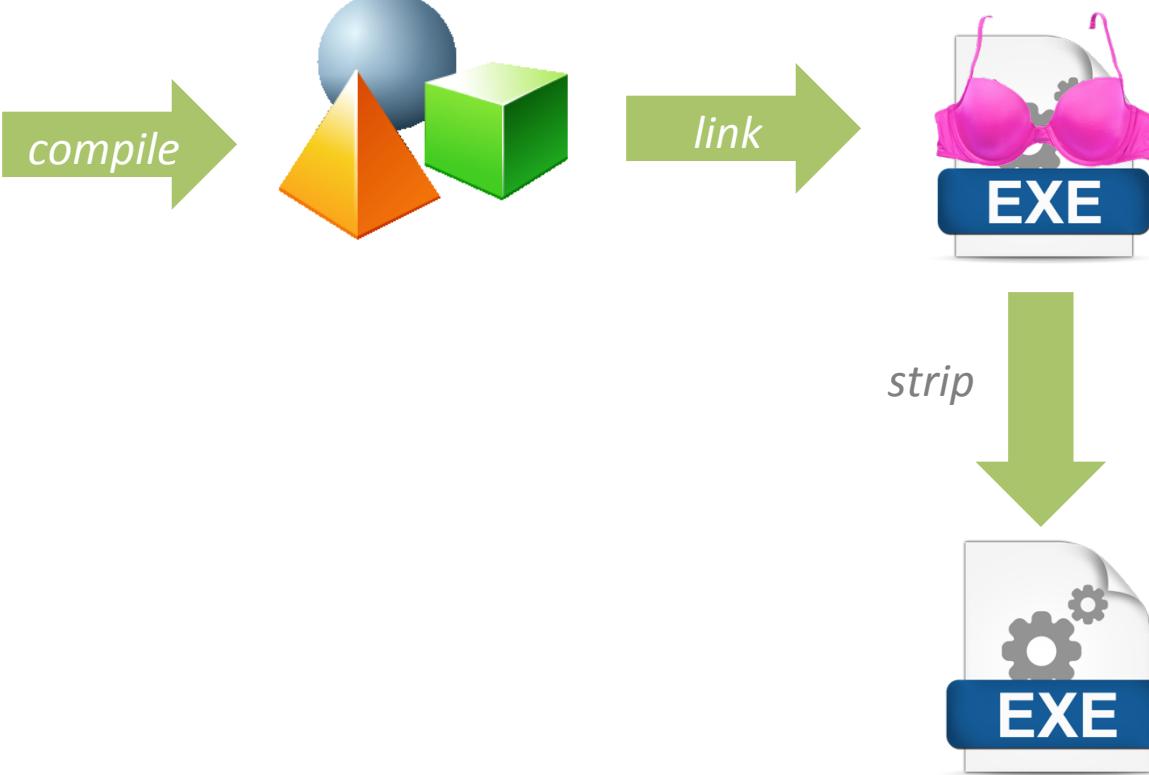
Challenges of Static Binary Analysis

- Get the binary code
- Binaries lack significant information present in source
- Often no clear library or operating system abstractions
 - where to start the analysis from?
 - hard to handle environment interactions

From Source to Binary Code

```
/// <summary>
/// Does a remote HTTP request, returning a XsdDocument containing the result.
/// It is meant to be used to get RSS feeds or other XML based responses.
/// </summary>
/// <param name="URL">A URL pointing to a page which returns XML</param>
/// <returns>A XsdDocument with the response from the URL or a DOM element with Exception as XML</returns>
public XsdDocument remoteHTTP(string URL)
{
    XsdDocument xsdDoc = new XsdDocument();
    try
    {
        System.Net.HttpWebRequest myRequest = (System.Net.HttpWebRequest)WebRequest.Create(URL);
        myRequest.OriginalUri = "Windows-2003-Printerm1.0 (MSIE 7.0; Windows NT 5.1)";
        System.Net.WebResponse myResponse = myRequest.GetResponse();
        Stream xsdStream = myResponse.GetResponseStream();
        xsdDoc.Load(xsdStream);
    }
    catch (Exception e)
    {
        XsdDataSection StackTraceCDATA = xsdDoc.CreateDataSection(e.StackTrace);
        XsdDataSection MessageCDATA = xsdDoc.CreateDataSection(e.Message);
        XsdDataSection SourceCDATA = xsdDoc.CreateDataSection(e.Source);
        XsdDataSection URLCDATA = xsdDoc.CreateDataSection(URL);
        XsdDataException xsdException = xsdDoc.CreateNode(XsdNodeType.Element, "Exception", "");
        XsdDataMessage xsdMessage = xsdDoc.CreateNode(XsdNodeType.Element, "Message", "");
        XsdDataSource xsdSource = xsdDoc.CreateNode(XsdNodeType.Element, "Source", "");
        XsdDataStackTrace xsdStackTrace = xsdDoc.CreateNode(XsdNodeType.Element, "StackTrace", "");
        XsdDataURL xsdURL = xsdDoc.CreateNode(XsdNodeType.Element, "URL", "");

        StackTrace.AppendChild(StackTraceCDATA);
        Message.AppendChild(MessageCDATA);
        Source.AppendChild(SourceCDATA);
        Exception.AppendChild(StackTrace);
        Exception.AppendChild(Message);
        Exception.AppendChild(Source);
        Exception.AppendChild(URL);
        xsdDoc.AppendChild(Exception);
    }
}
return xsdDoc;
}
```

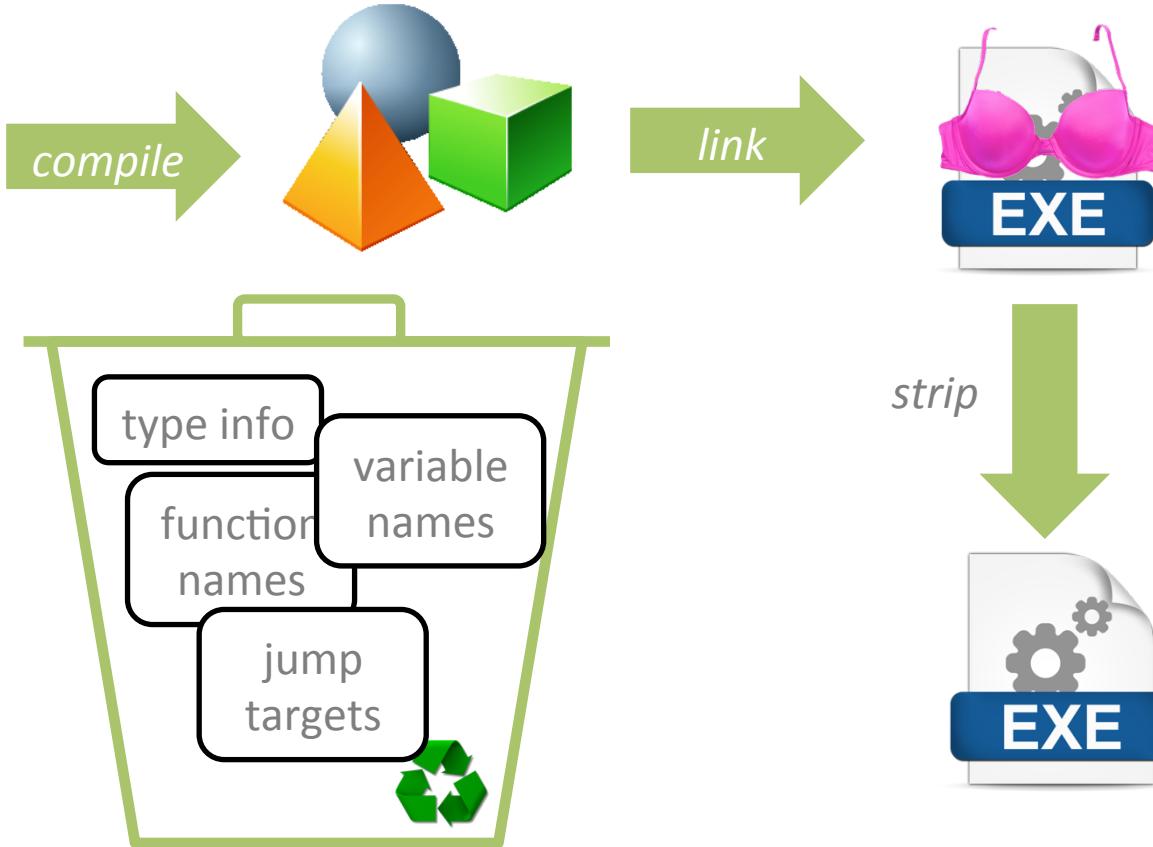


From Source to Binary Code

```

/// <summary>
/// Does a remote HTTP request, returning a XsdDocument containing the result.
/// It is meant to be used to get RSS feeds or other XML based responses.
/// </summary>
/// <param name="URL">A URL pointing to a page which returns XML</param>
/// <returns>A XsdDocument with the response from the URL or a DOM element with Exception as XML</returns>
public XsdDocument remoteHTTP(string URL)
{
    XsdDocument xsdDoc = new XsdDocument();
    try
    {
        System.Net.HttpWebRequest myRequest = (HttpWebRequest)WebRequest.Create(URL);
        myRequest.OriginalUri = "Windows-2003-Printers\\1.0 (MSIE 7.0; Windows NT 5.1)";
        System.Net.WebResponse myResponse = myRequest.GetResponse();
        Stream readStream = myResponse.GetResponseStream();
        xsdDoc.Load(readStream);
    }
    catch (Exception e)
    {
        XsdCataction StackTraceCATA = xsdDoc.CreateCataction(e.StackTrace);
        XsdCataction MessageCATA = xsdDoc.CreateCataction(e.Message);
        XsdCataction URLCATA = xsdDoc.CreateCataction(URL);
        XsdNode Exception = xsdDoc.CreateNodes(XsdNodeType.Element, "Exception", "");
        XsdNode Message = xsdDoc.CreateNode(XsdNodeType.Element, "Message", "");
        XsdNode Source = xsdDoc.CreateNode(XsdNodeType.Element, "Source", "");
        XsdNode StackTrace = xsdDoc.CreateNode(XsdNodeType.Element, "StackTrace", "");
        StackTrace.AppendChild(StackTraceCATA);
        Message.AppendChild(MessageCATA);
        Exception.AppendChild(Source);
        Exception.AppendChild(StackTrace);
        Exception.AppendChild(Message);
        Exception.AppendChild(Source);
        Exception.AppendChild(URL);
        xsdDoc.AppendChild(Exception);
    }
    return xsdDoc;
}

```



Missing OS and Library Abstractions

- (Linux) system call interface is great
 - you know what the I/O routines are
 - important to understand what user can influence
 - you have typed parameters and return values
 - let's the analysis focus on (much smaller) main program
- OS is not there or embedded in binary blob
 - heuristics to find I/O routines
 - open challenge to find mostly independent components

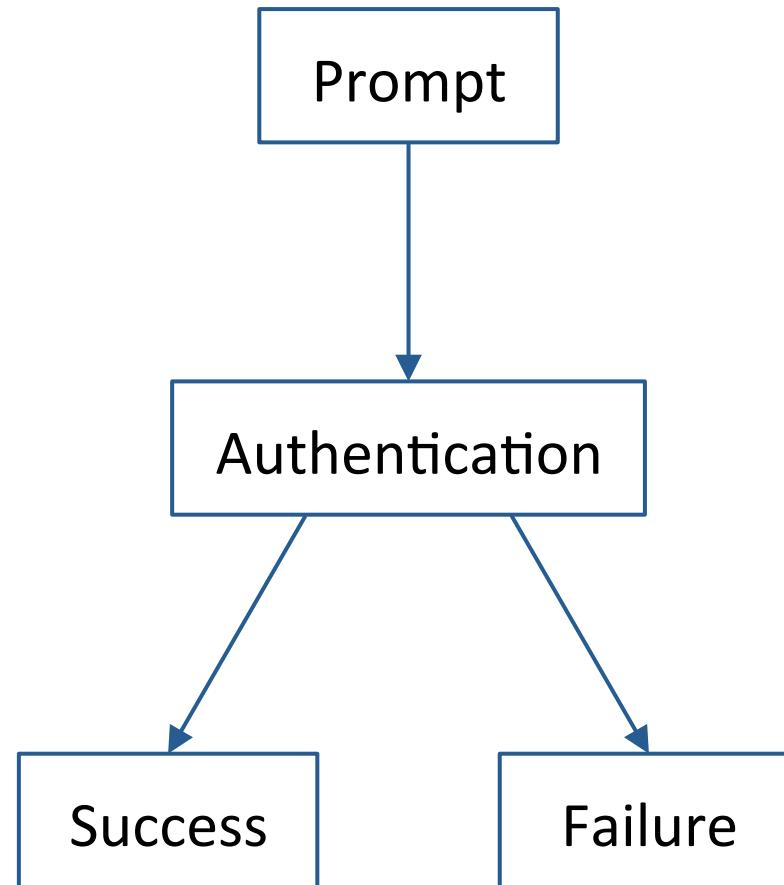
Missing OS and Library Abstractions

- Library functions are great
 - you know what they do and can write a “function summary”
 - you have typed parameters and return values
 - let’s the analysis focus on (much smaller) main program
- Library functions are embedded (like static linking)
 - need heuristics to rediscover library functions
 - IDA FLIRT (Fast Library Identification and Recognition Technology)
 - more robustness based on looking for control flow similarity

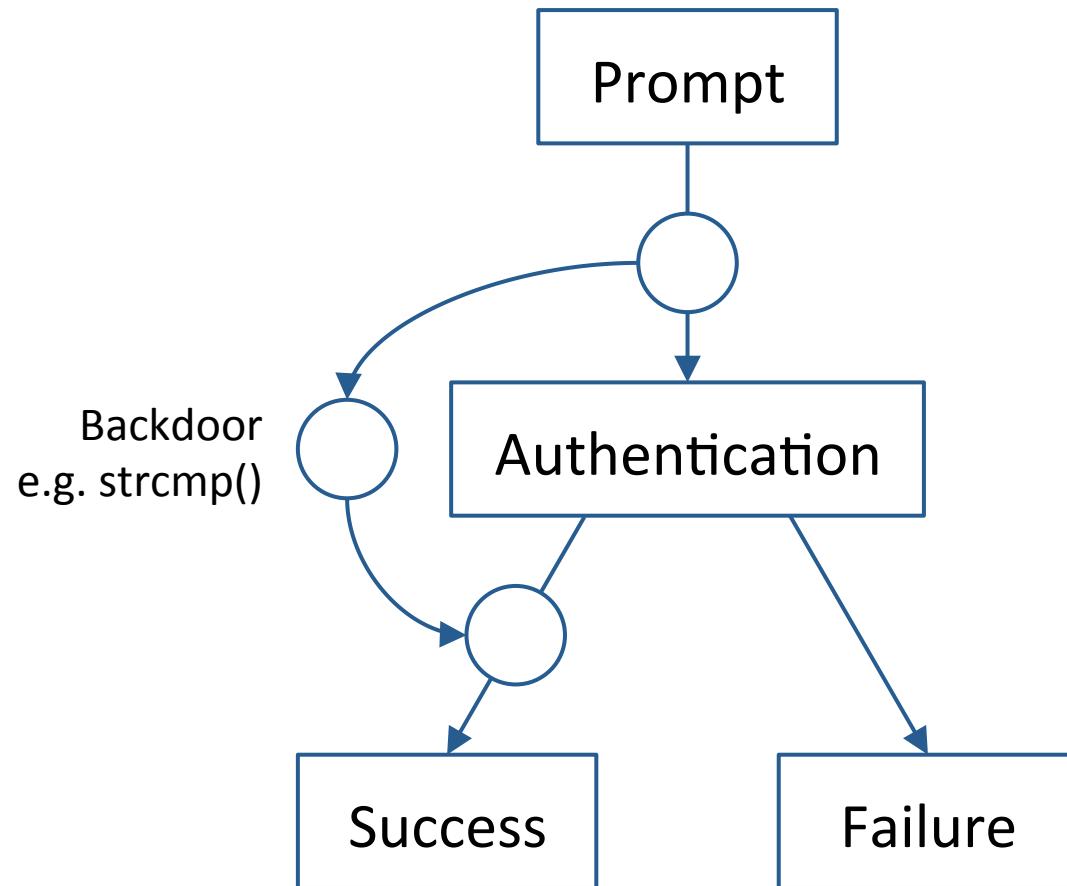
Types of Vulnerabilities

- Memory safety vulnerabilities
 - buffer overrun
 - out of bounds reads (heartbleed)
 - write-what-where
- Authentication bypass (backdoors)
- Actuator control!

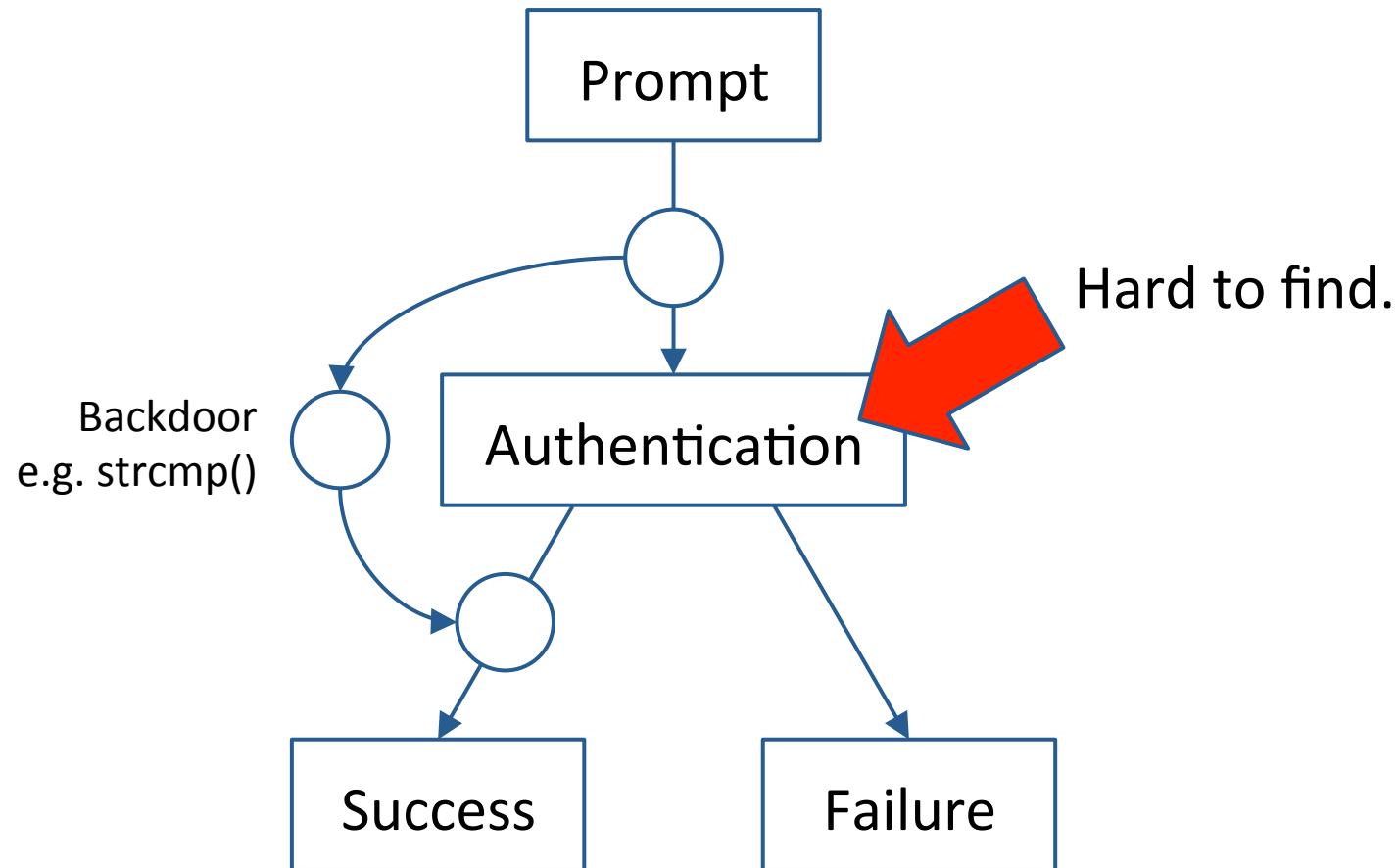
Authentication Bypass



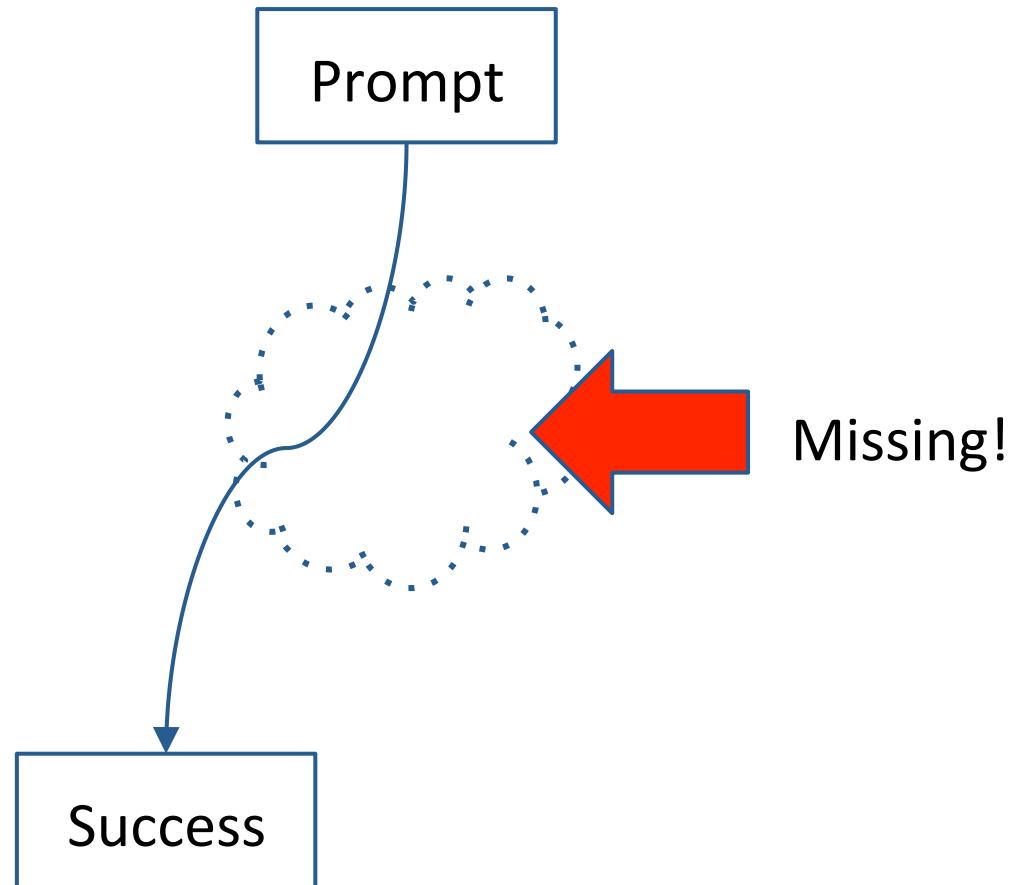
Authentication Bypass



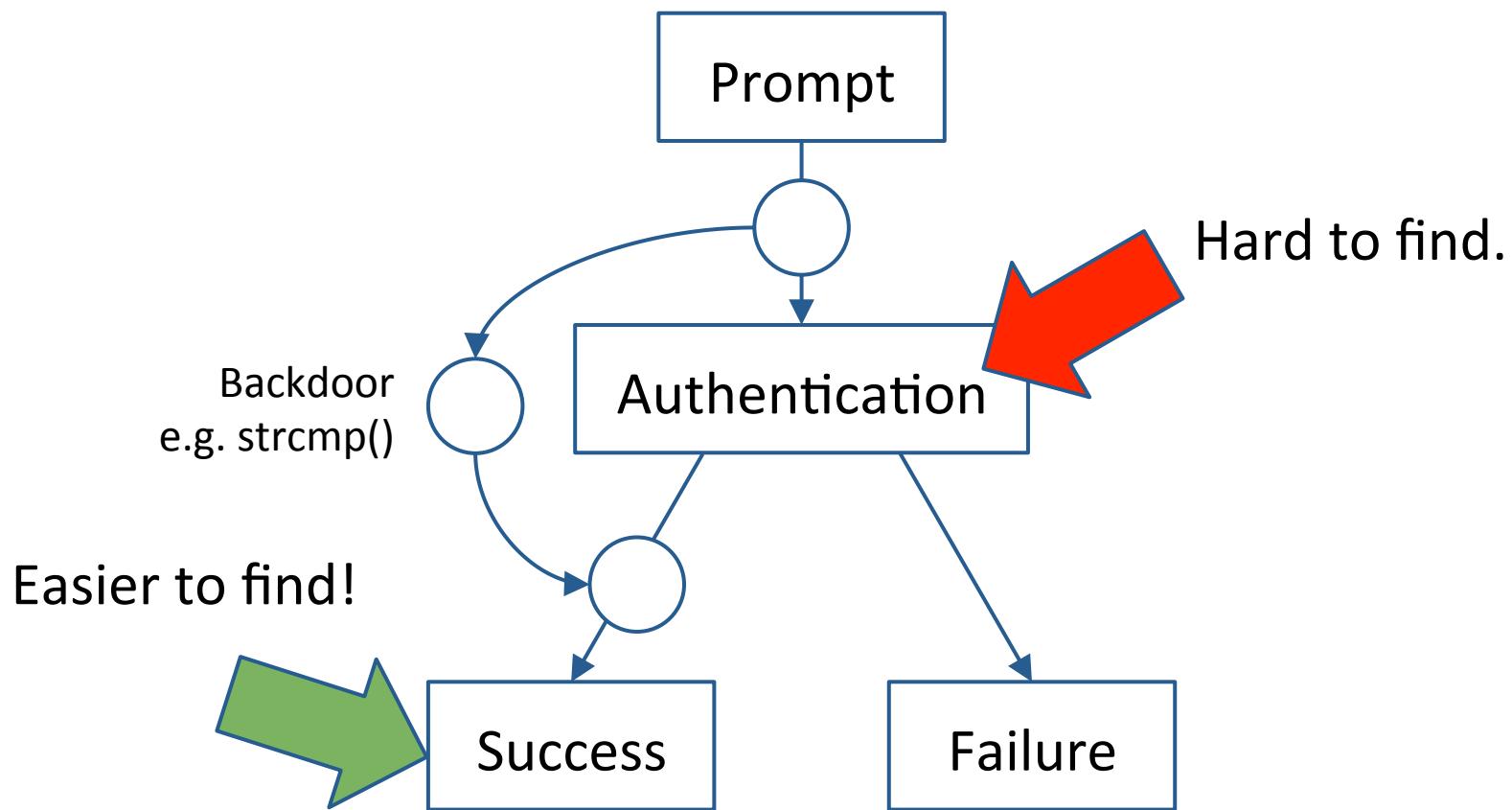
Authentication Bypass



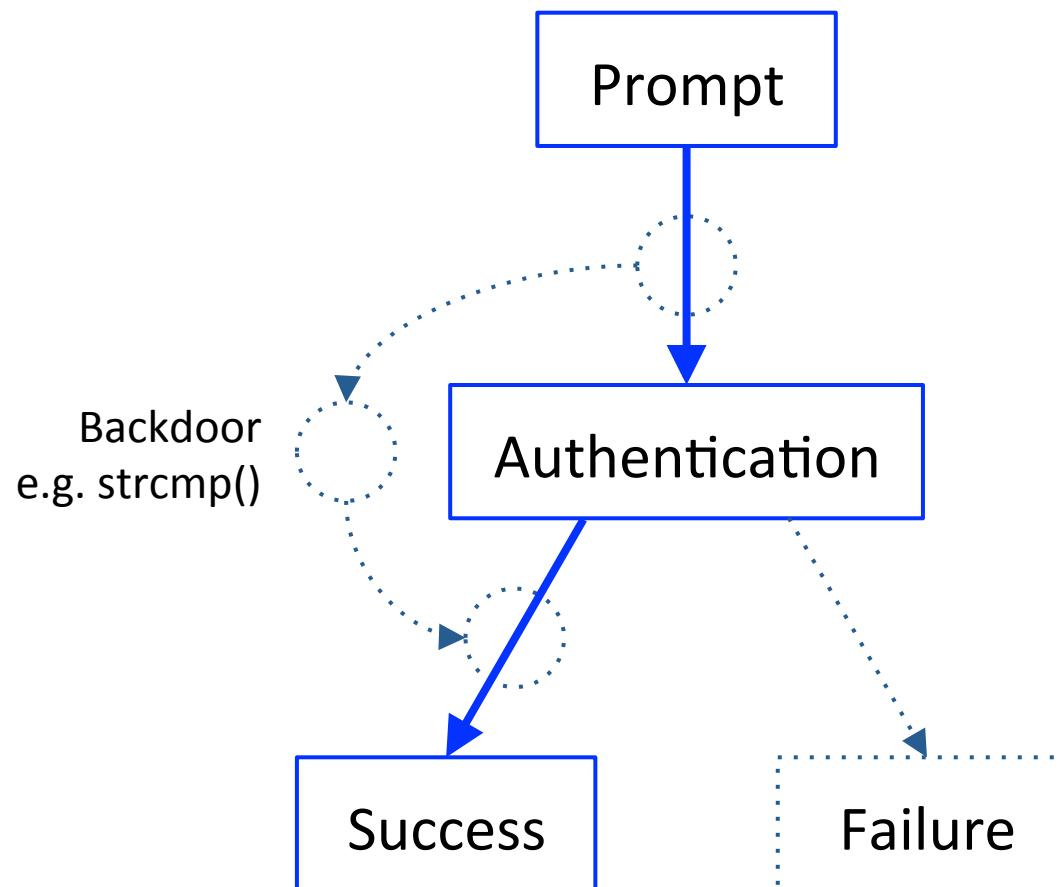
Authentication Bypass



Modeling Authentication Bypass



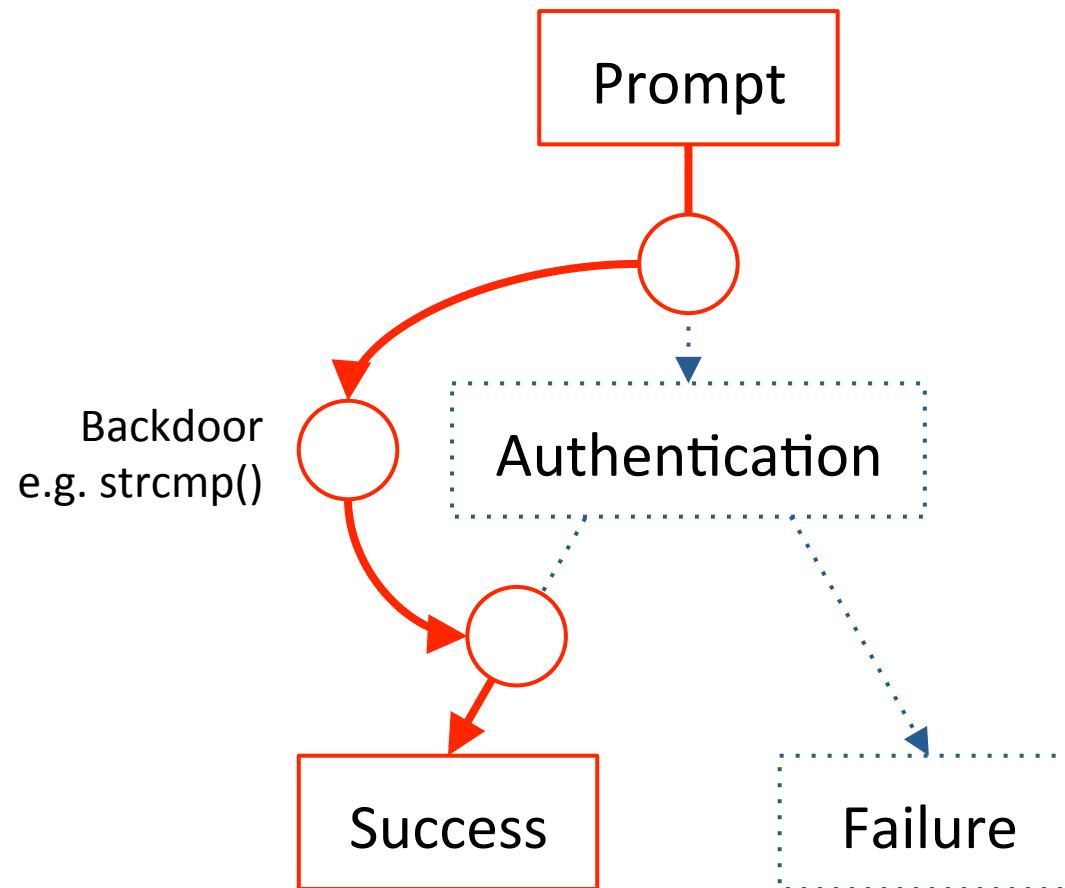
Input Determinism



Can we determine the input needed to reach the success function, just by analyzing the code?

The answer is NO

Input Determinism

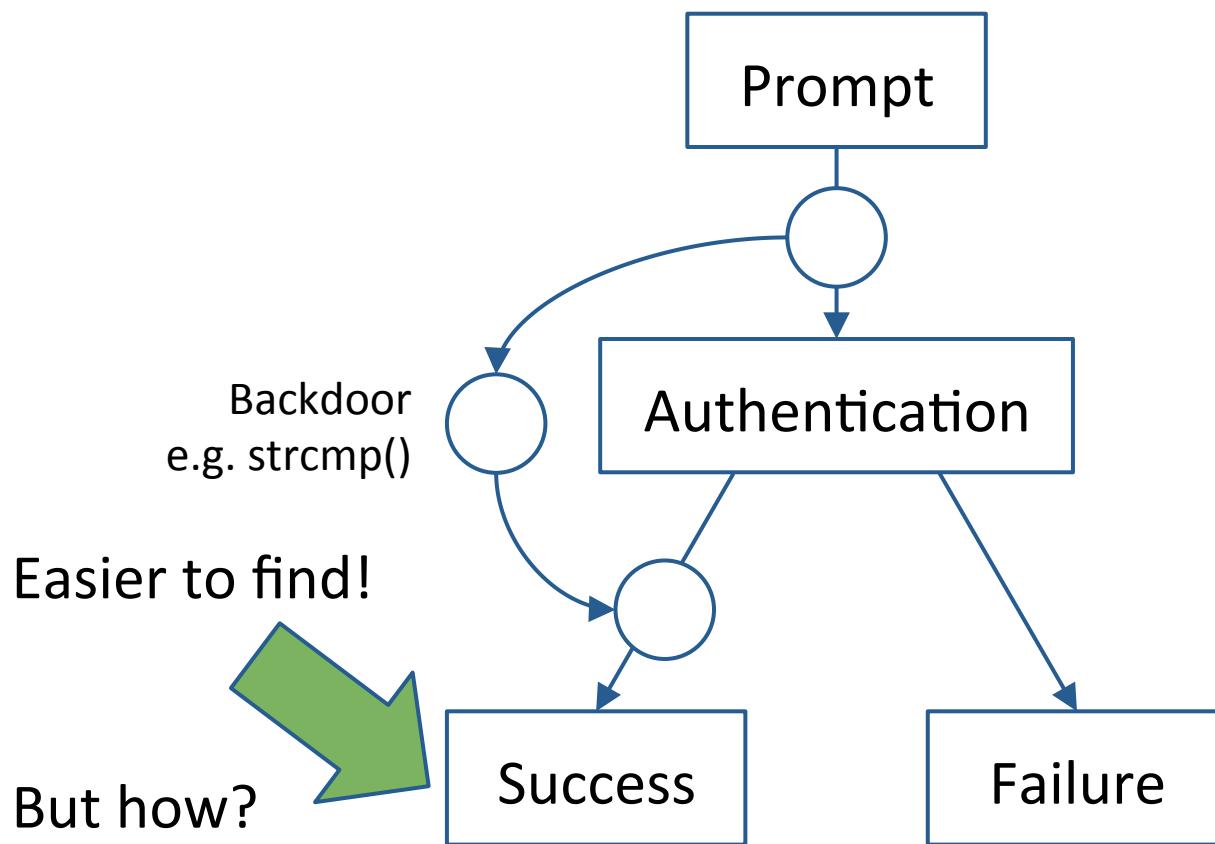


Can we determine the input needed to reach the success function, just by analyzing the code?

The answer is YES



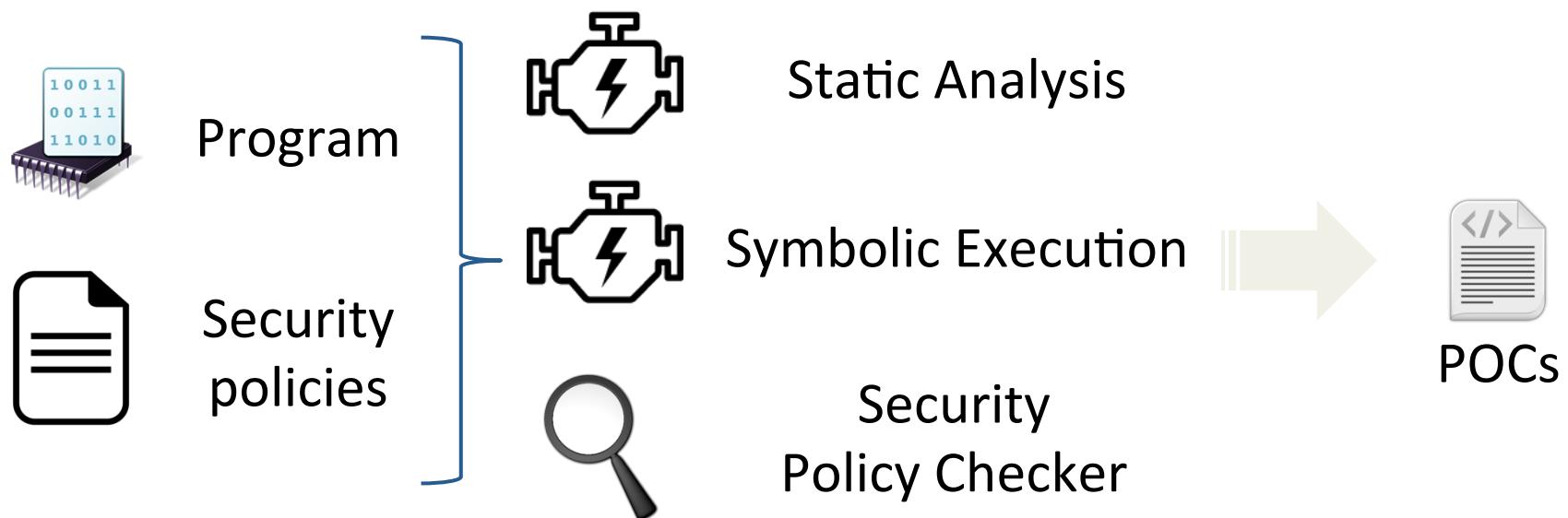
Modeling Authentication Bypass



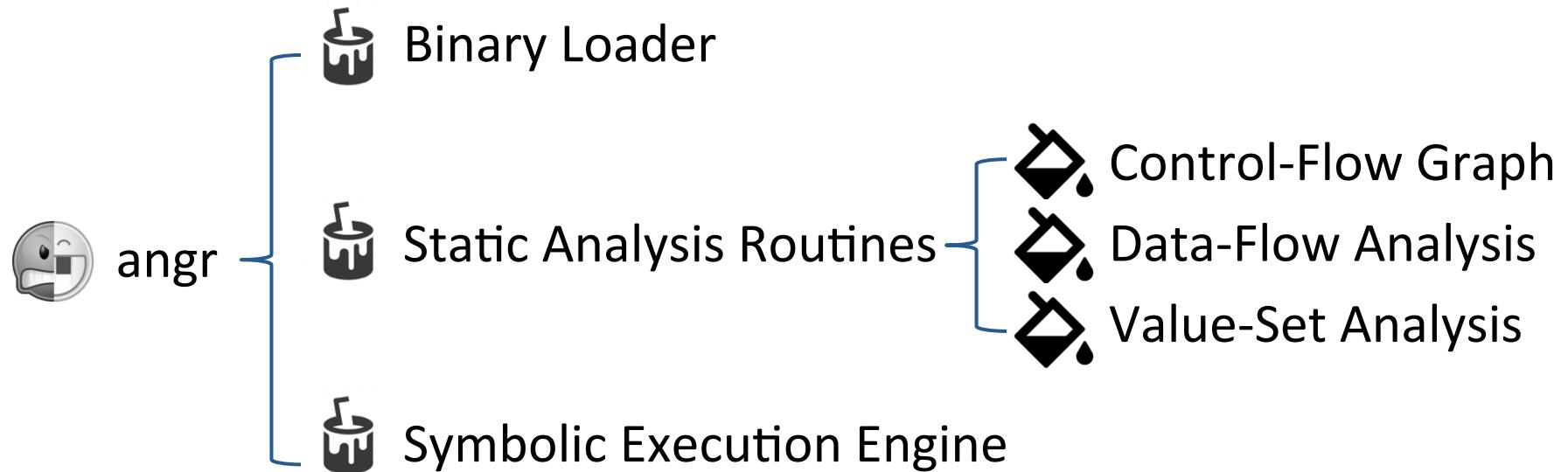
Finding “Authenticated Point”

- Without OS/ABI information:
 - Manual reverse engineering
 - Program outputs/references certain strings (like “welcome admin”)
 - Program accesses sensitive memory regions
- With ABI information:
 - Program calls sensitive syscalls
 - Program accesses sensitive resources/files

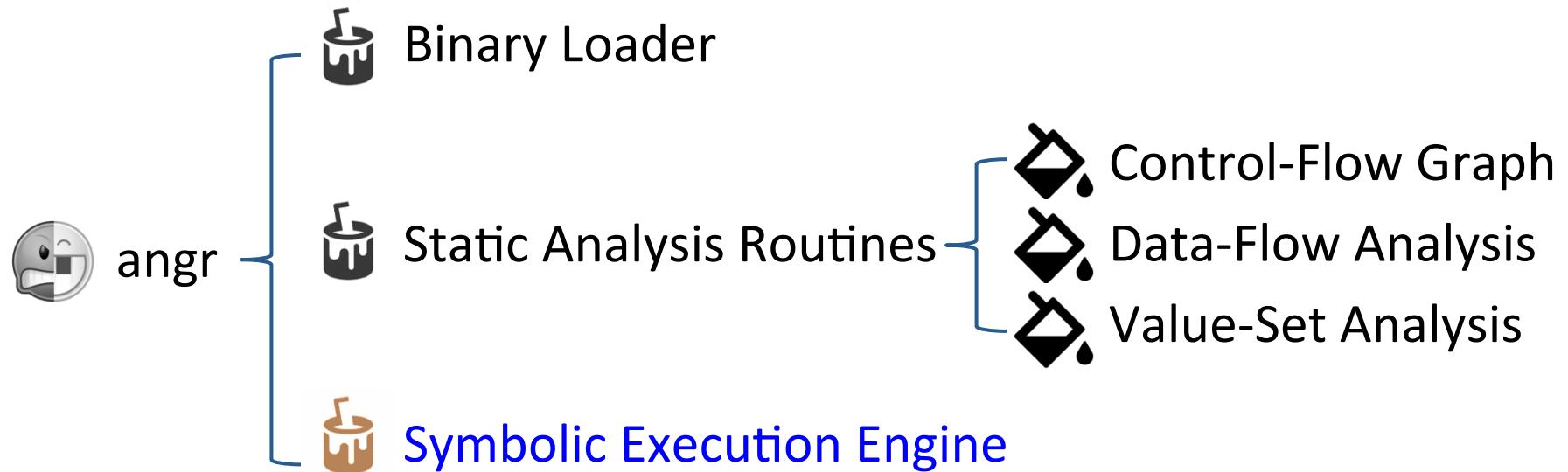
Using angr to Hunt for Vulnerabilities



angr: A Binary Analysis Framework



angr: A Binary Analysis Framework



Symbolic Execution

"How do I trigger path X or condition Y?"

- Dynamic analysis
 - Input A? No. Input B? No. Input C? ...
 - Based on concrete inputs to application.
- (Concrete) static analysis
 - "You can't"/"You might be able to"
 - Based on various static techniques.

We need something slightly different.

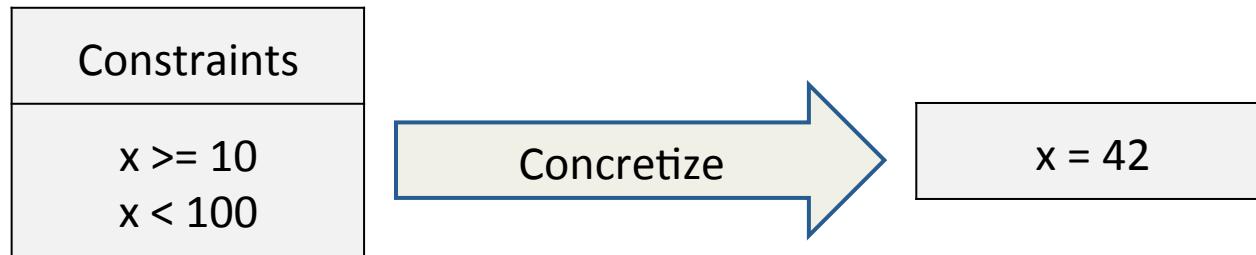


Symbolic Execution

"How do I trigger path X or condition Y?"

1. Interpret the application.
2. Track "constraints" on variables.
3. When the required condition is triggered,
"concretize" to obtain a possible input.

Symbolic Execution



Constraint solving:

- ❑ Conversion from set of constraints to set of concrete values that satisfy them.
- ❑ NP-complete, in general.



Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```

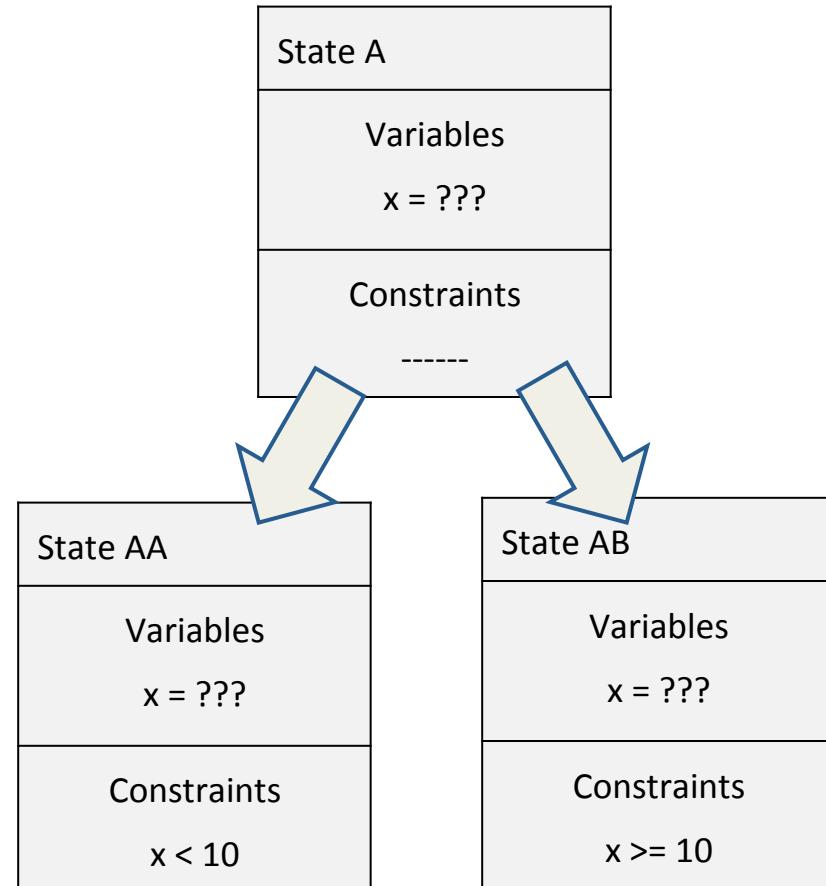
Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```

State A
Variables
x = ???
Constraints

Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```



Symbolic Execution

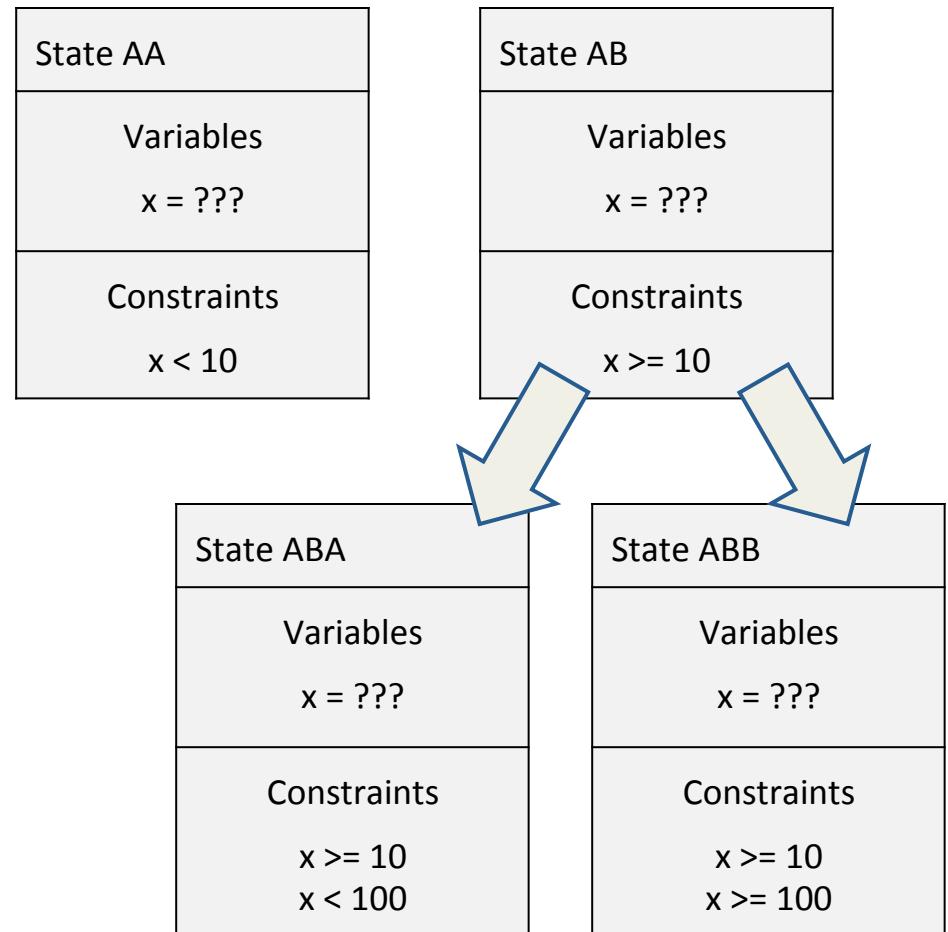
```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```

State AA
Variables $x = ???$
Constraints $x < 10$

State AB
Variables $x = ???$
Constraints $x \geq 10$

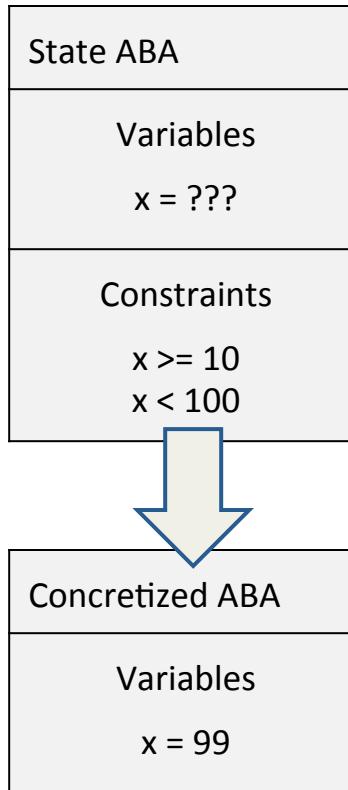
Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```



Symbolic Execution

```
x = int(input())
if x >= 10:
    if x < 100:
        print "Two!"
    else:
        print "Lots!"
else:
    print "One!"
```



Symbolic Execution - Pros and Cons

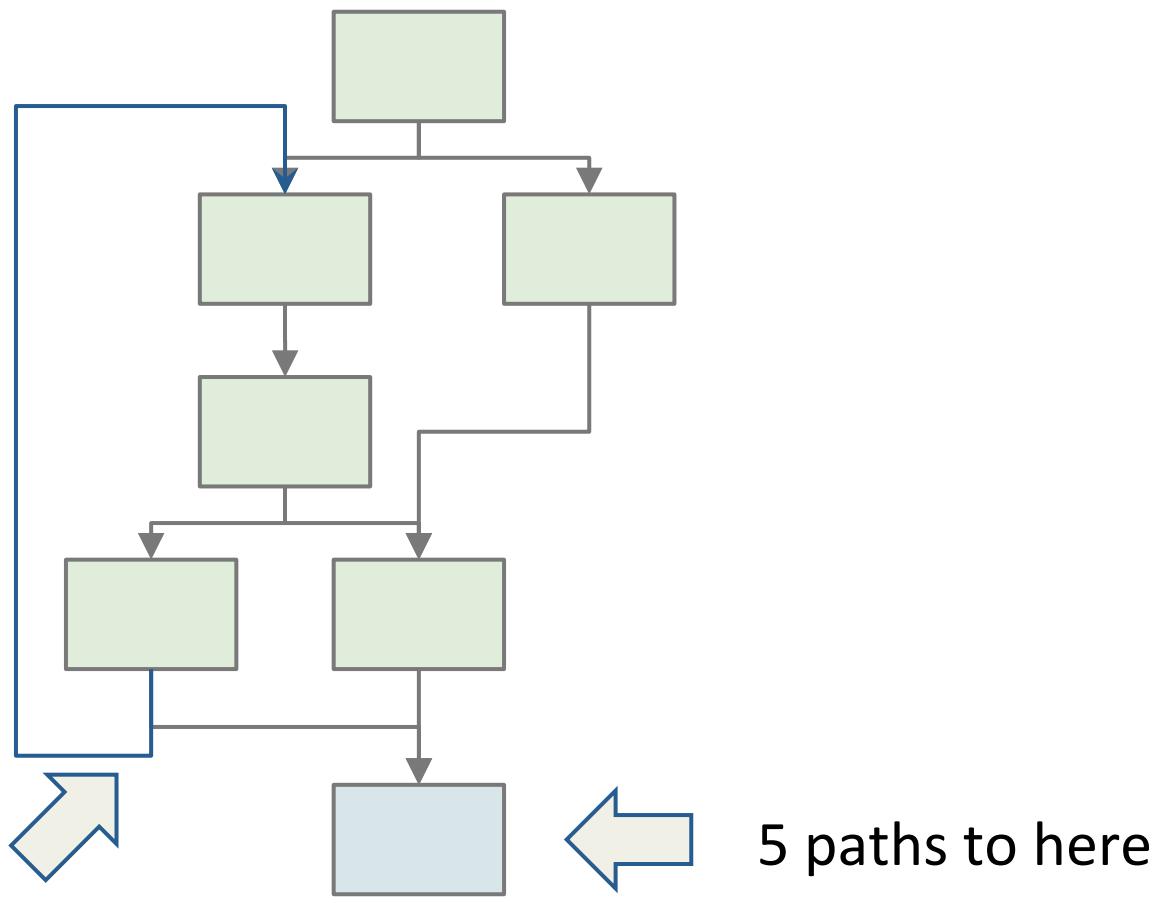
Pros

- Precise
- No false positives (with correct environment model)
- Produces directly-actionable inputs

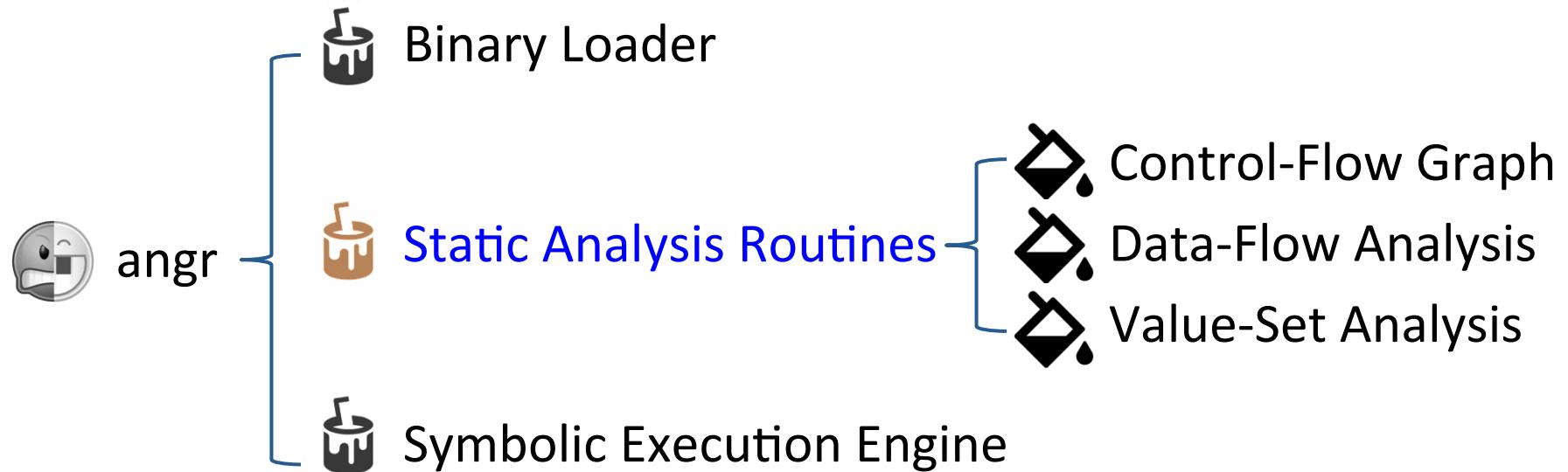
Cons

- Not scalable
- constraint solving is np-complete
- path explosion

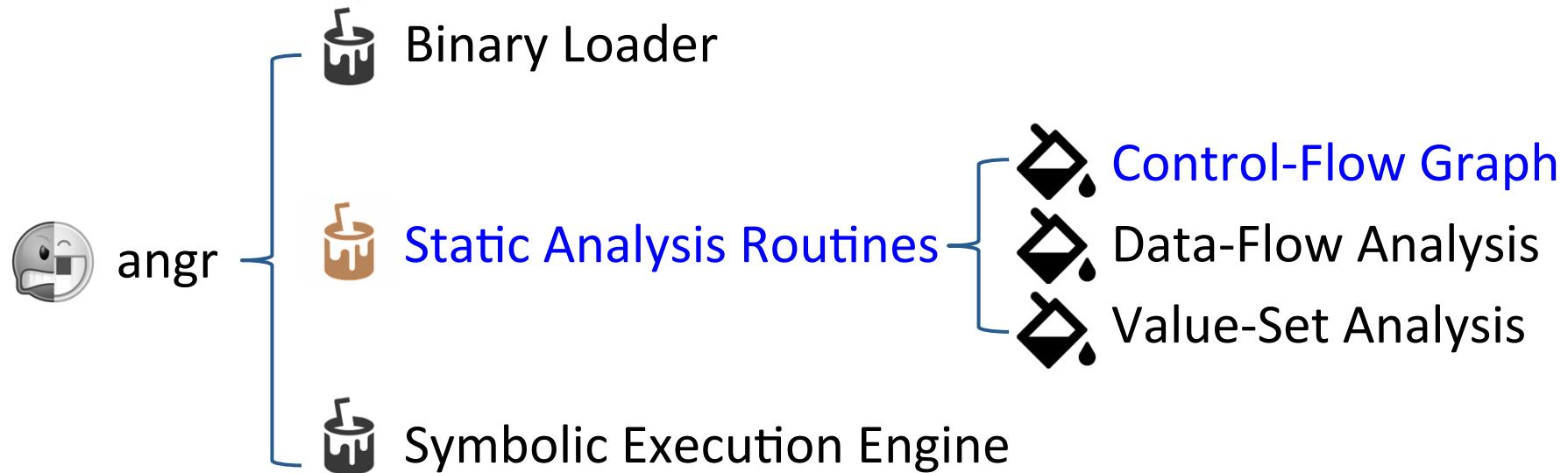
Path Explosion



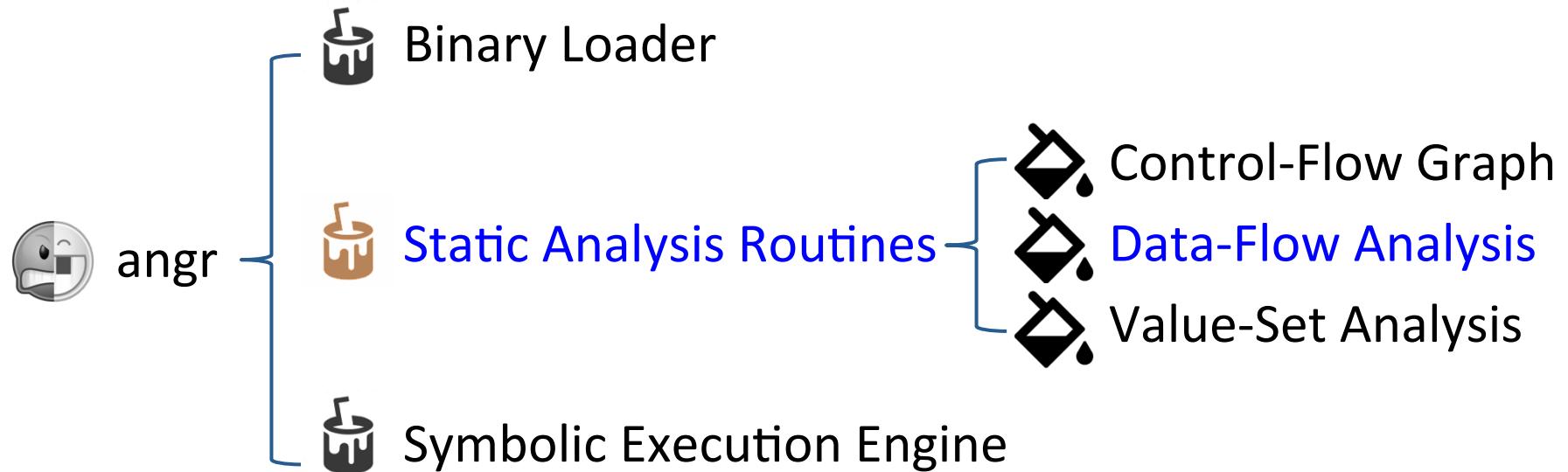
angr: A Binary Analysis Framework



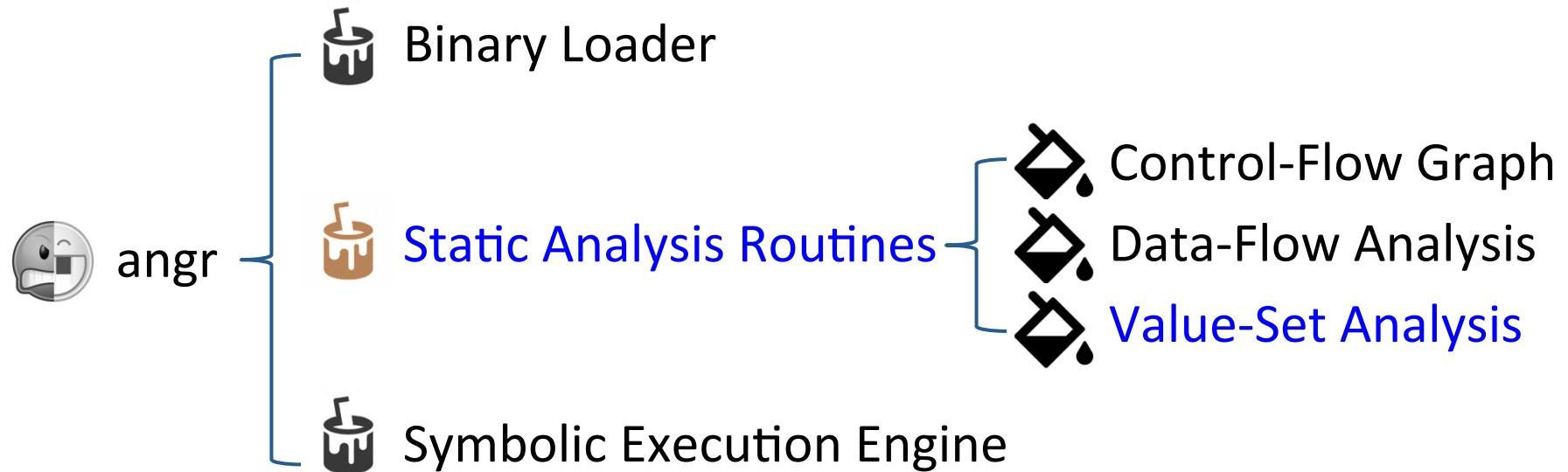
angr: A Binary Analysis Framework



angr: A Binary Analysis Framework



angr: A Binary Analysis Framework



Value Set Analysis

Memory access checks

Type inference

Variable recovery

Range recovery

Wrapped-interval analysis

Value-set analysis

Abstract interpretation

Value Set Analysis

```
{  
    ( global,  
      (4[0x601000, 0x602000],  
       32) ),  
    ( stack_0x400957, (8[-0xc, -0x4],  
      32) )  
}
```

global
...
0x601000, 0x601004
0x601008, 0x60100c
...

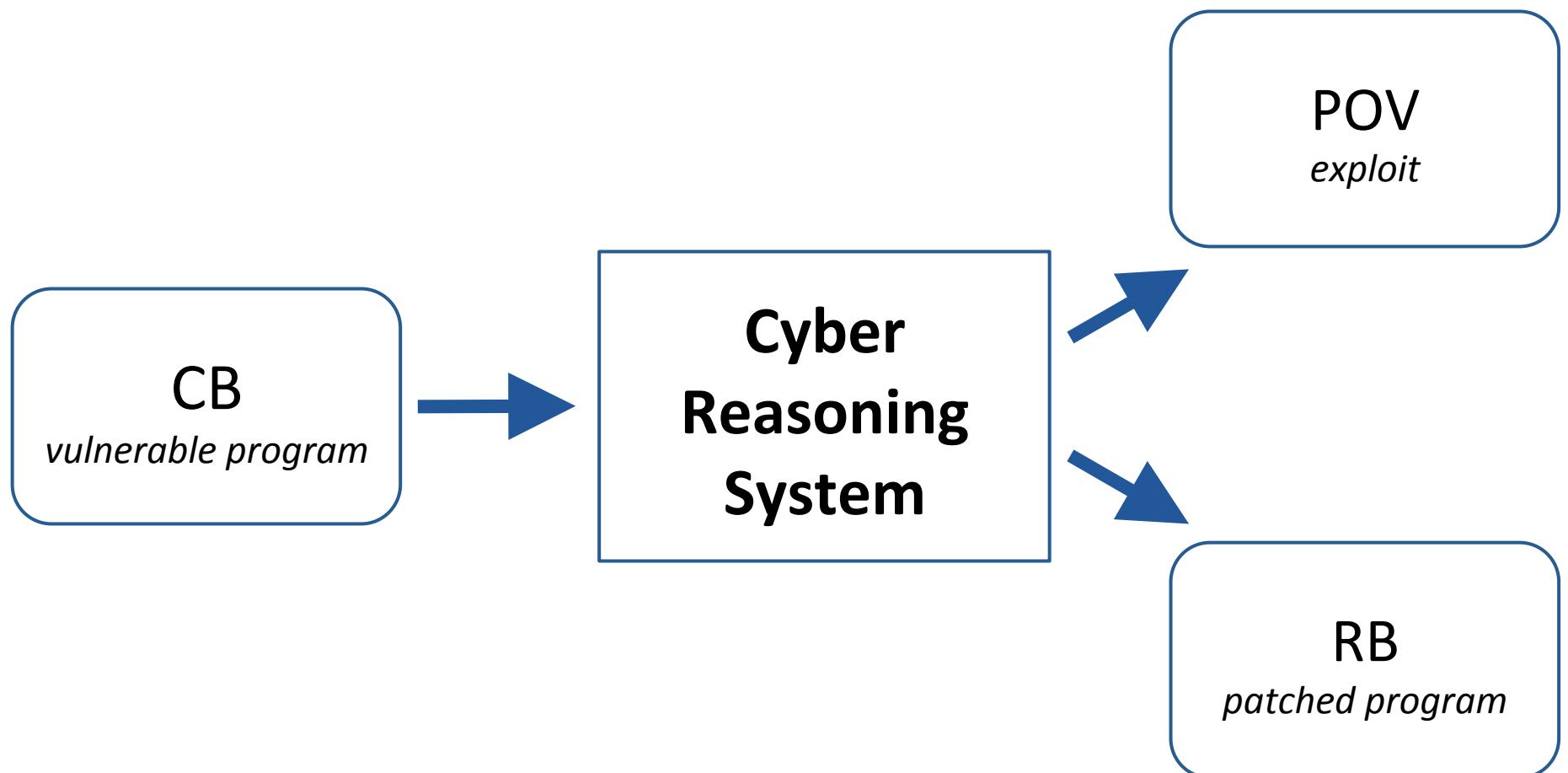
stack_0x400957
...
- 0xc
- 0x4
...

What have we used this for?

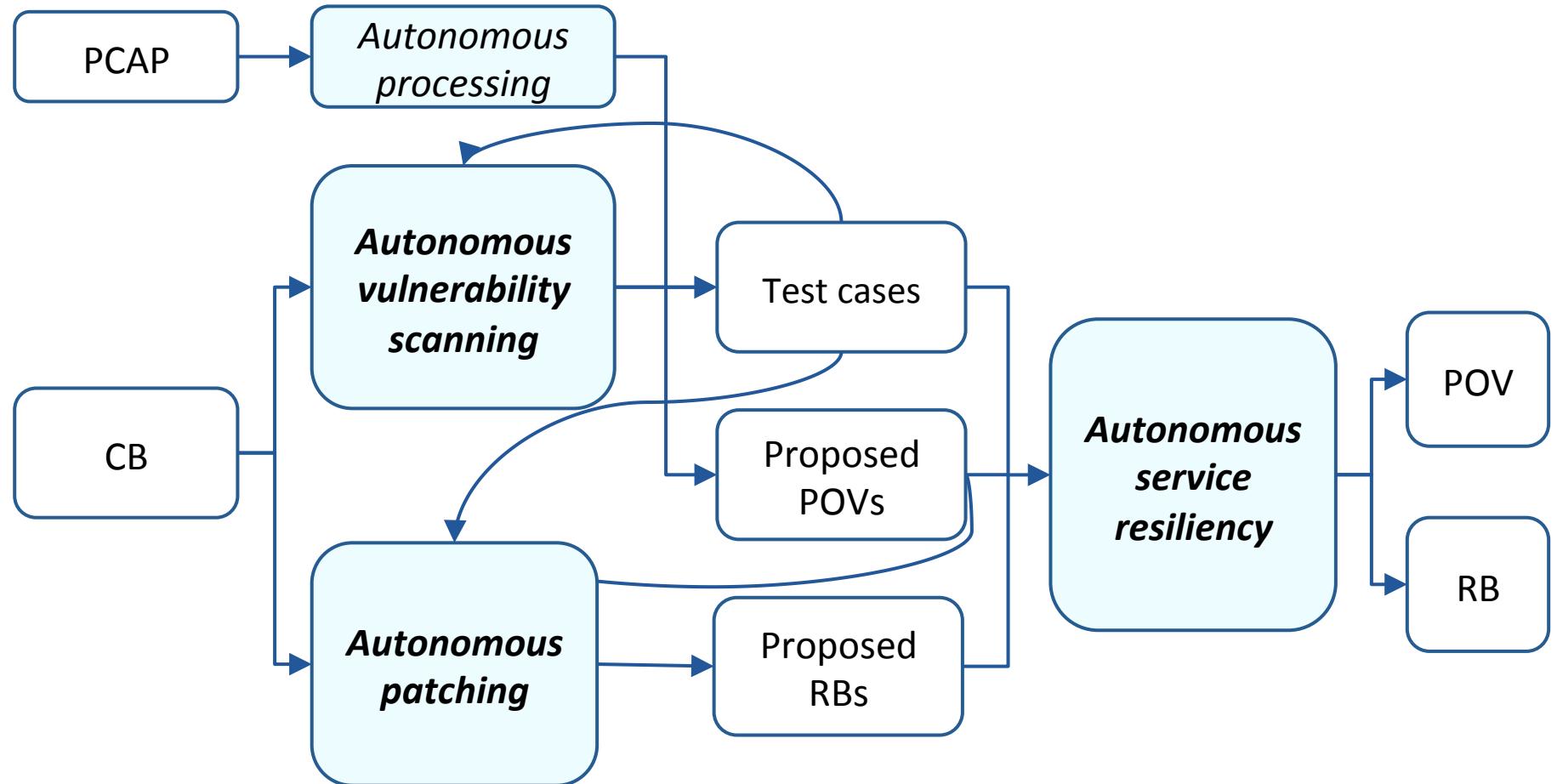


- angr
 - ─ Binary Loader
 - ─ Static Analysis Routines
 - ─ Control-Flow Graph
 - ─ Data-Flow Analysis
 - ─ Value-Set Analysis
 - ─ Symbolic Execution Engine

The Cyber Grand Challenge!



The Shellphish CRS



The Shellphish CRS

