Binary Static Analysis

Chris Wysopal, CTO and Co-founder

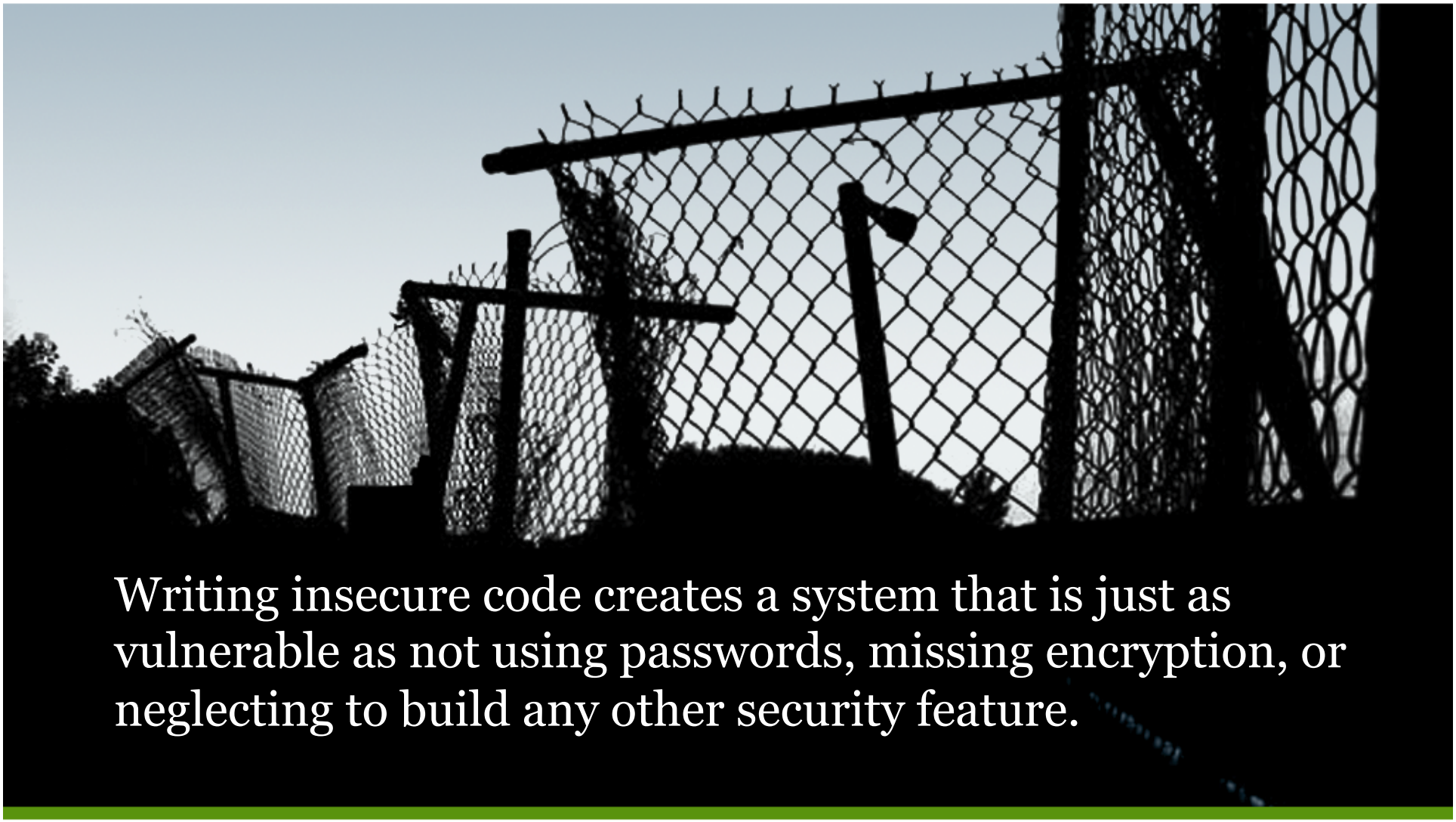
March 7, 2012

Introduction to Computer Security - COMP 116

Bio

Chris Wysopal, Veracode's CTO and Co-Founder, is responsible for the company's software security analysis capabilities. In 2008 he was named one of InfoWorld's Top 25 CTO's and one of the 100 most influential people in IT by eWeek. In 2010, he was named a SANS Security Thought Leader.

In the 90's he was one of the original vulnerability researchers at The L0pht. He has testified on Capitol Hill in the US on the subjects of government computer security and how vulnerabilities are discovered in software. Chris Wysopal is the lead author of "The Art of Software Security Testing" published by Addison-Wesley.

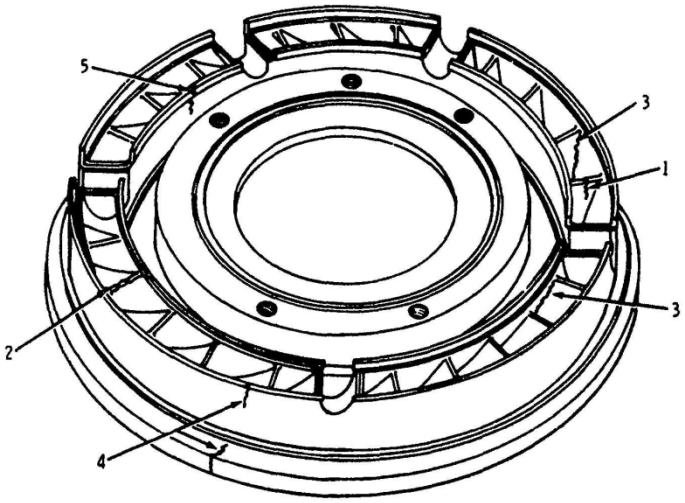

Writing insecure code creates a system that is just as vulnerable as not using passwords, missing encryption, or neglecting to build any other security feature.

Evolution of Computer Intrusions

- ✓Misconfiguration of networks or hosts
  - ✓ Weak or blank passwords, world readable file shares
- ✓Vulnerability in underlying OS or other "infrastructure" hardware/software
  - ✓ Steady stream of updates from Microsoft, IBM, Cisco, Oracle
- ✓"Social Engineering" or tricking the user
  - ✓ Download and run malicious codec or install free AV, phishing, clicking on link to exploit client software vulnerabilities
- ✓Vulnerabilities in software
  - ✓ Media players, desktop software, web applications

Static Analysis

- ✓ Analysis of software performed without actually executing the program
- ✓ Full coverage of the entire source or binary
- ✓ In theory, having full application knowledge can reveal a wider range of bugs and vulnerabilities than the “trial and error” of dynamic analysis
- ✓ Impossible to identify vulnerabilities based on system configuration that exist only in the deployment environment



Benefits of Binary Static Analysis

- ✓ C99 specification has many unspecified, or implementation defined, constructs, e.g.:
  - ✓ Order of function argument evaluation

```
int i = 0;
foo(i++, i++);       // foo(0, 1), or foo(1, 0)
bar(a(), b(), c());  // where a(), b(), and c() have side effects
```

  - ✓ Order of expressions

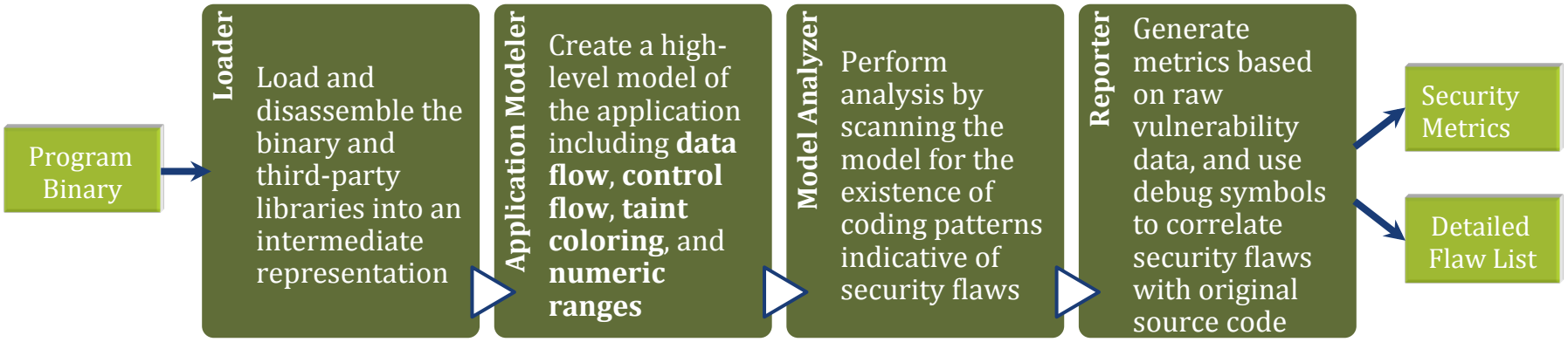
```
int i = 0;
a[++i] = i;          // a[1] = 1, or a[1] = 0
```

  - ✓ Many others (Google for ”nasal demons”)
- ✓ Detect flaws in third-party libraries
- ✓ Is the compiler trustworthy? (Ken Thompson, “Reflections on Trusting Trust”)

Benefits Of Binary Analysis

- ✓ Binary analysis has 100% coverage. All code can be analyzed, regardless of source availability.
- ✓ Exact modeling of control flow in the presence of compiler switches is automatic, for example, buffer checks do not have to be emulated.
- ✓ You analyze exactly what is being shipped. Backdoors inserted in source, compiled, and then removed from source will still be found.
- ✓ Binary-level flaws such as optimizations that remove memory clearing of cryptographic keys, can be detected.
- ✓ Code is always analyzed in its complete execution context. Analyzing 'pieces' of programs leads to higher false-positive rates.

Binary Static Analysis Architecture



Components of Static Binary Analysis

- ✓Binary Modeler
  - ✓This component builds a model from the binary directly,

    producing a high-level representation of the program that includes reconstructed dataflow and control flow elements suitable for human consumption or machine-based inspection.
- ✓Intermediate Representation
  - ✓This component is the core of the analysis, the data structure that represents the entire 'meaning' of the program being analyzed, designed carefully to represent everything and make assumptions about very little to nothing. There are some liberties one can take here but you have to be very careful!

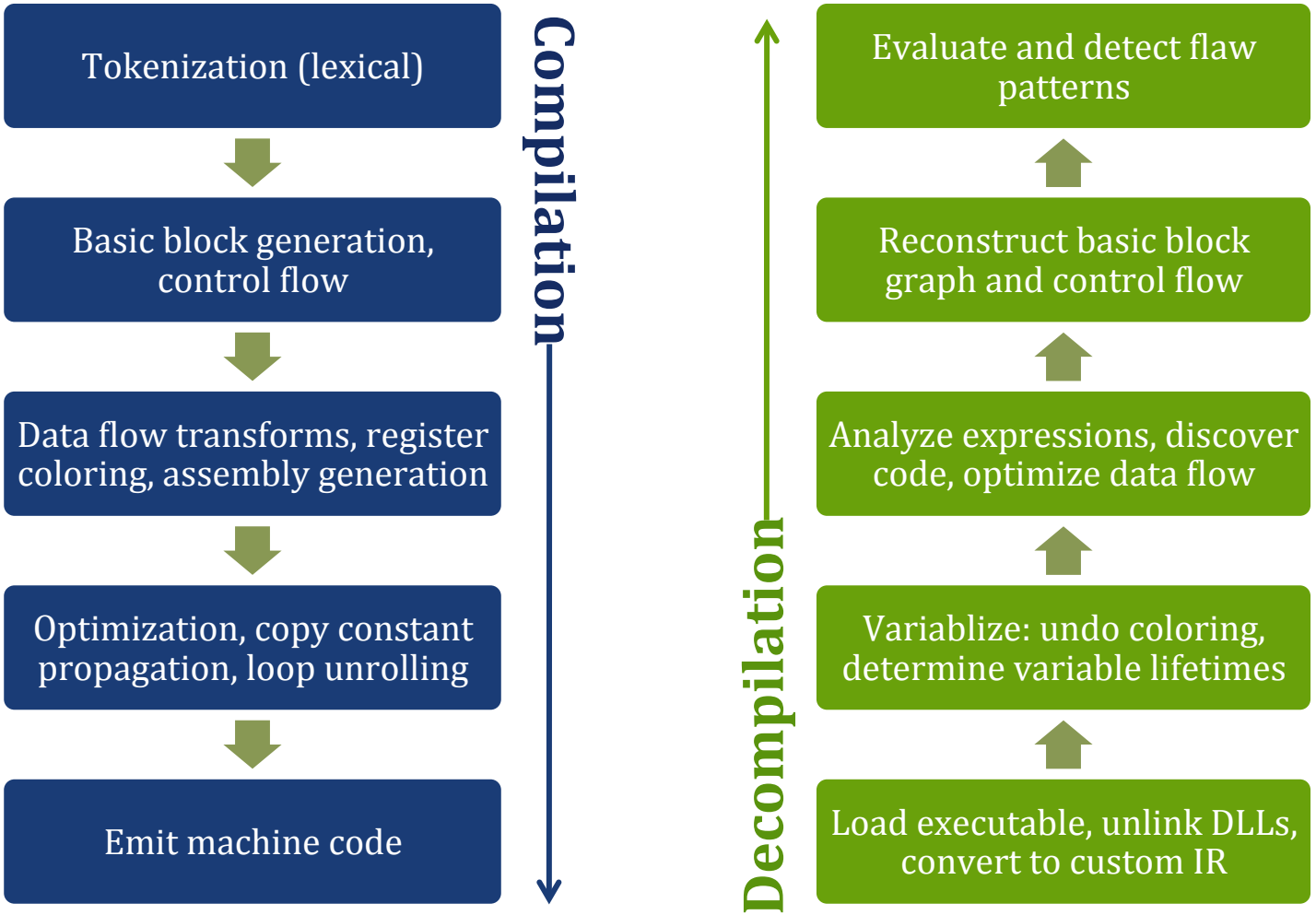
Program Structure (SOM)

- ✓Describes how the program is organized
  - ✓Procedures and functions
  - ✓Libraries
  - ✓Class layout
  - ✓Data structure layout
- ✓Not much analysis performed here, but provides the foundation for data flow and control flow phases to be layered on

Components of Static Binary Analysis

- ✓Model Querying and Condition Searching System
  - ✓This is responsible for searching the intermediate model for characteristics.
  - ✓In the case of Veracode, this is the part that looks for ‘security flaws’. At no point in the process before this stage does ‘security’ really come into play.
  - ✓Static Binary Analysis could easily be looking for other things such as general code quality problems or to compare two models for equivalent pieces via graph isomorphism that might suggest code being stolen and reused elsewhere

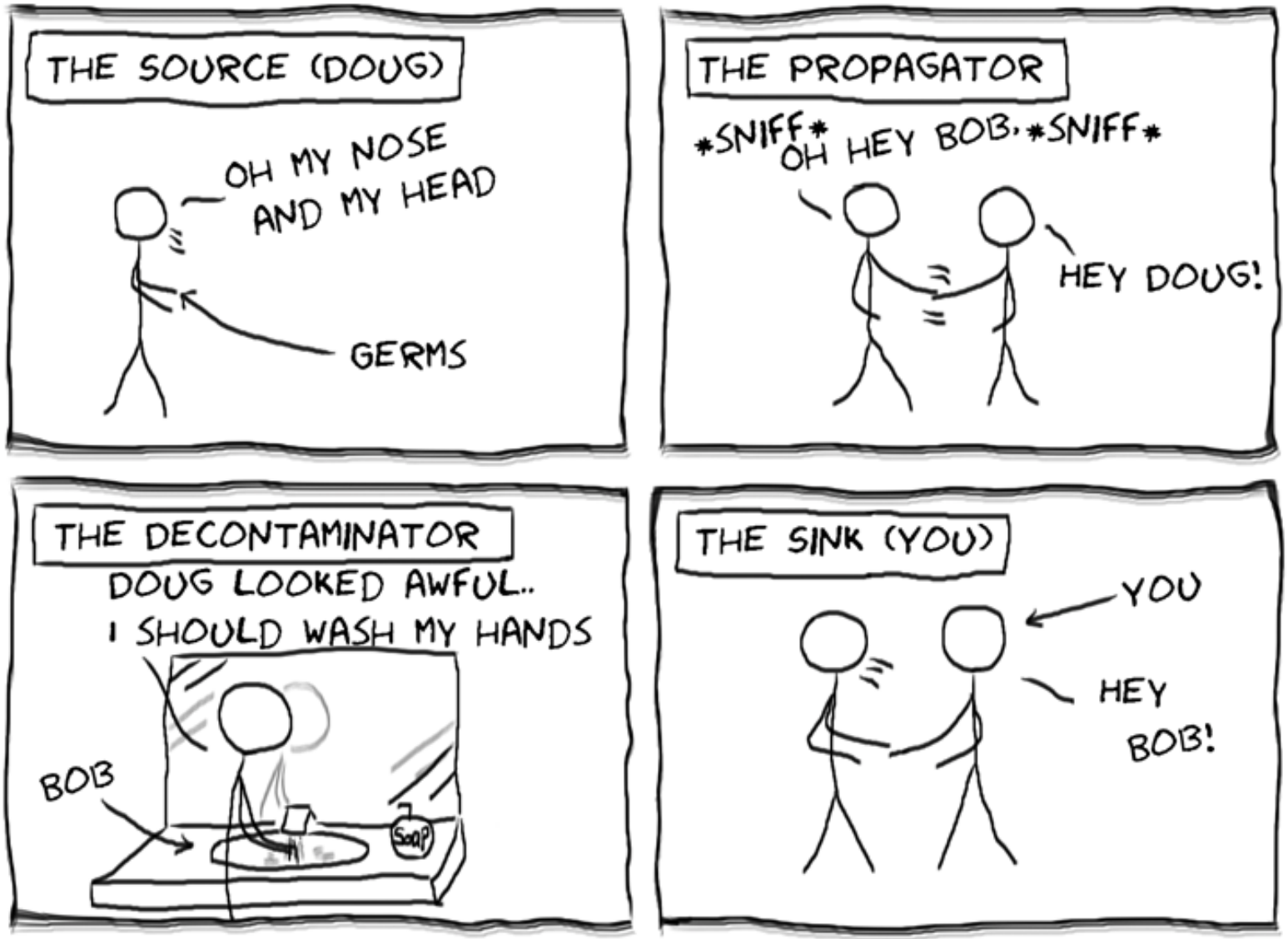
Decompilation is Compilation in Reverse



VERACODE

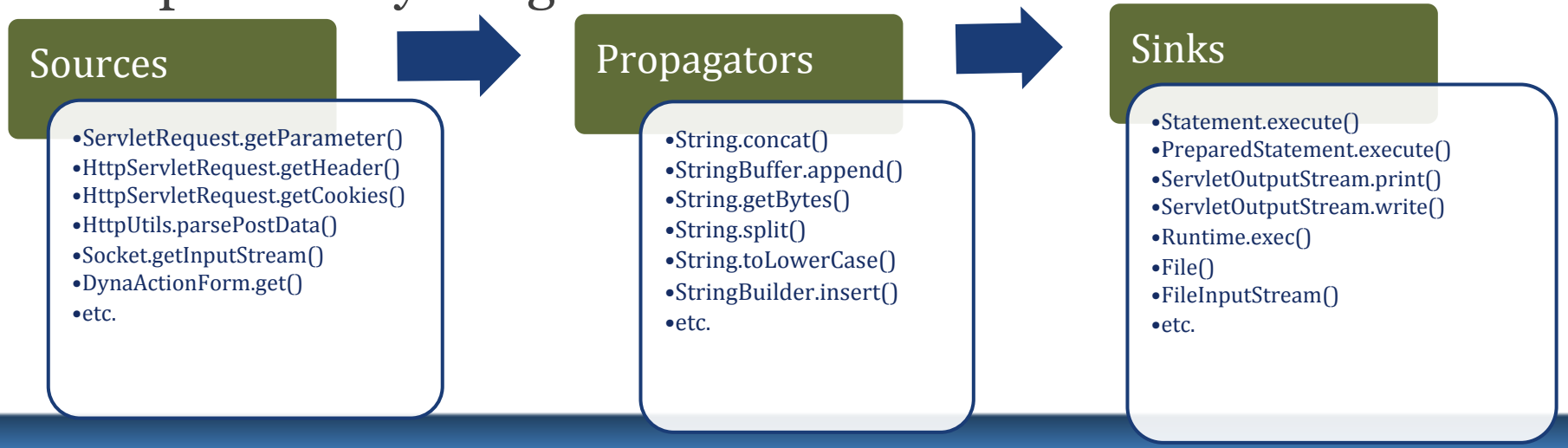
Veracode doesn't decompile to source but rather makes use of decompiler concepts to build our control flow and data flow models

A Brief Explanation of Taint



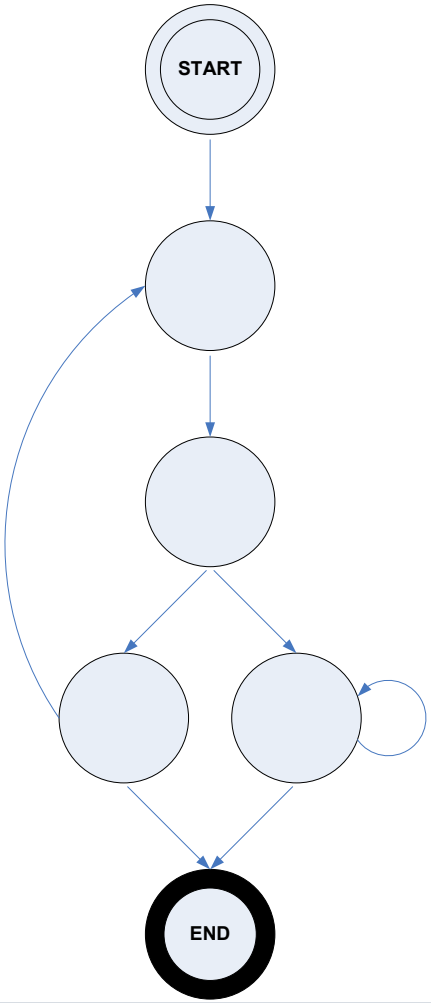
Data Flow

- ✓Track data from the time it enters the program throughout its variable lifetime
- ✓Multiple taint colors can be applied to any piece of data: untrusted, sensitive, decrypted, fromstorage, fromnetwork, etc.
- ✓Report locations where tainted data is used in a potentially dangerous situation



Control Flow

- A **basic block** is code that has one entry point, one exit point and no jump instructions contained within it
- A **control flow graph** represents all paths that might be traversed during execution; it is a group of basic blocks with directed edges
- Consider **virtual function calls**; what appears to be a simple call to myFunction() may actually have 10 different control flow edges



Numeric Ranges

- ✓ Attempt to predict the range of values for a variable at a particular location in the code
- ✓ Use these ranges in conjunction with type information, buffer sizes, etc. to detect memory corruption issues and other numeric flaws

```
char lookup(int idx) {
    char buf[32];
    load_values(buf);
    return buf[idx];
}
```

Q: What is the range of idx?
A: INT_MIN to INT_MAX

```
char lookup_safe(int idx) {
    char buf[32];
    load_values(buf);
    if (idx < 32) {
        return buf[idx];
    } else {
        printf("idx was %d", idx);
        return 0;
    }
}
```

Q: What is the range of idx?
A: INT_MIN to 31
(oops, still vulnerable)

Q: What is the range of idx?
A: 32 to INT_MAX

Veracode Query Language (VQL)

- ✓Declarative scan language
- ✓Abstracts away internals
- ✓Makes scan intent clear
- ✓Makes scan reviewable
- ✓Shortens development time

```
scan TimeBomb1 {
   match {
     L1: now = time( _ );
     L2: VQL_COMPARE( now, bombtime );
   }
   where {
     AlwaysConst( bombtime, L2 );
   }
   perform {
     Annotate( L2, VULN_Time_Bomb );
   }
}
```

Detecting Flaws

- ✓Coding flaws can be represented by patterns
- ✓Pose a series of questions to the control flow and data flow models to determine if those patterns exist

- ✓***Example***: Is user-supplied data ever concatenated into an ad-hoc SQL query?
  - ✓Relies on control flow, data flow, untrusted taint color, and knowledge of database APIs

Detecting Flaws

- ***Example***: Is sensitive information ever exfiltrated from a mobile application?
  - Relies on control flow, data flow, sensitive taint color, and network communication APIs for a specific mobile platform

Questions?

Thank You!