BitBlaze: a New Approach for Computer Security via Binary Analysis

Dawn Song

Computer Science Dept.

UC Berkeley

Malicious Code---Critical Threat on the Internet

- **Diverse forms**
  - **Worms, botnets, spyware, viruses, trojan horses, etc.**
- **High prevelance**
  - **CodeRed Infected 500,000 servers**
  - **61% U.S. computers infected with spyware [National Cyber Security Alliance06]**
  - **Millions of computers in botnets**
- **Fast propagation**
  - **Slammer scanned 90% Internet within 10 mins**
- **Huge damage**
  - **$10billion annual financial loss [ComputerEconomics05]**

Defense is Challenging

- **Software inevitably has bugs/security vulnerabilities**
  - **Intrinsic complexity**
  - **Time-to-market pressure**
  - **Legacy code**
  - **Long time to produce/deploy patches**
- **Attackers have real financial incentives to exploit them**
  - **Thriving underground market**
- **Large scale zombie platform for malicious activities**
- **Attacks increase in sophistication**

- **We need more effective techniques and tools for defense**
  - **Previous approaches largely symptom & heuristics based**

The BitBlaze Approach

- **Semantics based, focus on root cause:**

  Automatically extracting security-related properties from binary code (vulnerable programs & malicious code) for effective defense

- **Automatically create high-quality detection & defense mechanisms**
  - **Automatic generation of vulnerability signatures to filter out exploits**
  - **Automatic detection and classification of malware**
    - » **Spyware, keylogger, rootkit, etc.**
  - **Automatic detection of botnet traffic**

- **Able to handle binary-only setting**
  - **Important for COTS & malicious code scenarios**
  - **Binary is truthful**

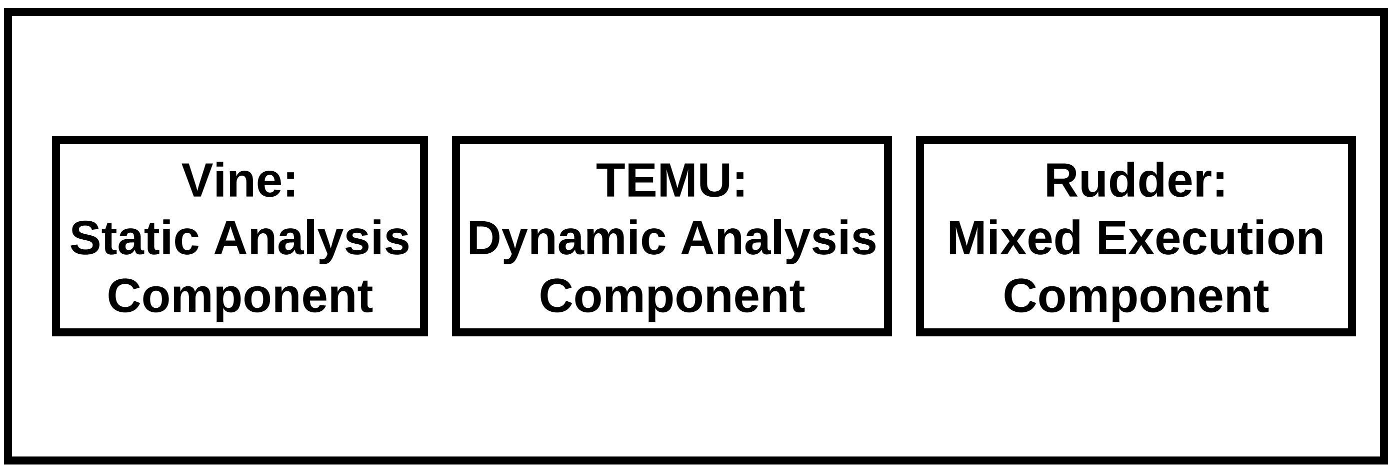
The BitBlaze Research Foci

1. **Design and develop a unified binary analysis platform for security applications**
   - **Identify & cater common needs of different security applications**
   - **Leverage recent advances in program analysis, formal methods, binary instrumentation/analysis techniques to enable new capabilities**

2. **Introduce binary-centric approach as a powerful arsenal to solve real-world security problems**
   - **COTS vulnerability discovery, diagnosis & defense**
   - **Malicious code analysis & defense**
   - **Automatic model extraction & analysis**
   - **More than a dozen security applications & publications**

BitBlaze Binary Analysis Infrastructure: Architecture

- **The first infrastructure:**
  - **Novel fusion of static, dynamic analysis techniques, and formal analysis techniques such as symbolic execution & abstract interpretation**
  - **Capable of analyzing whole system (including OS kernel)**
  - **Capable of analyzing packed/encrypted/obfuscated code**



BitBlaze Binary Analysis Platform

Outline

- **BitBlaze in action: sample security applications**
  - **Automatic patch-based exploit generation**
  - **Automatic Signature Generation**
  - **In-depth malware analysis**
- **BitBlaze Binary Analysis Infrastructure**
  - **Challenges**
  - **Design rationale**
  - **Architecture**

Patch Tuesday

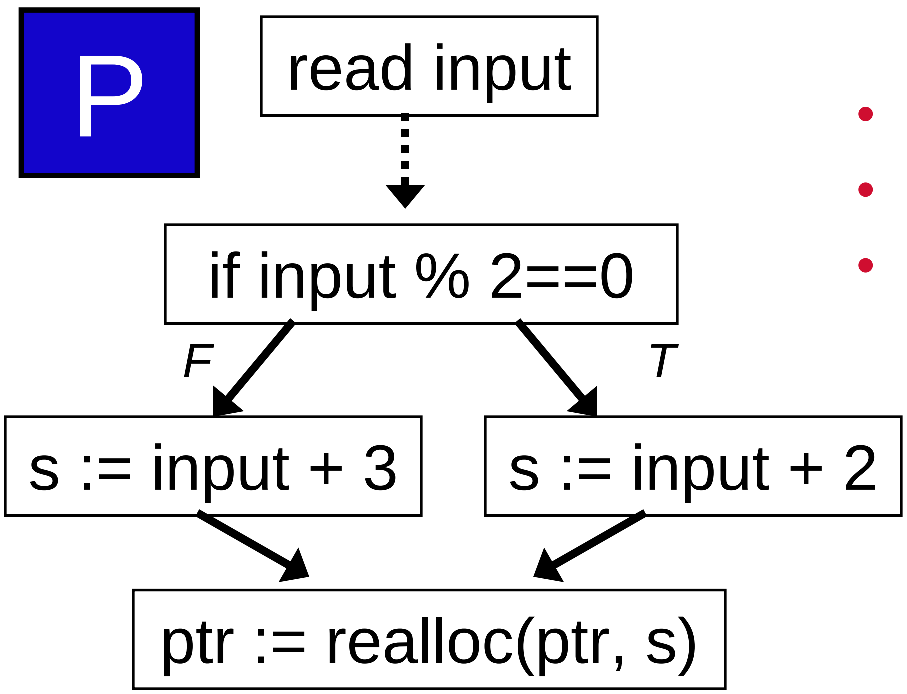
- **On the surface: security patches fix vulnerabilities**
- **Beneath the surface:**
  - **What's the security consequence of a patch release?**
- **Our work:**
  - **Current patch approach is dangerous**
  - **Automatic exploit generation**

Automatic Patch-based Exploit Generation

- **Given vulnerable program P, patched program P', automatically generate exploits for P**

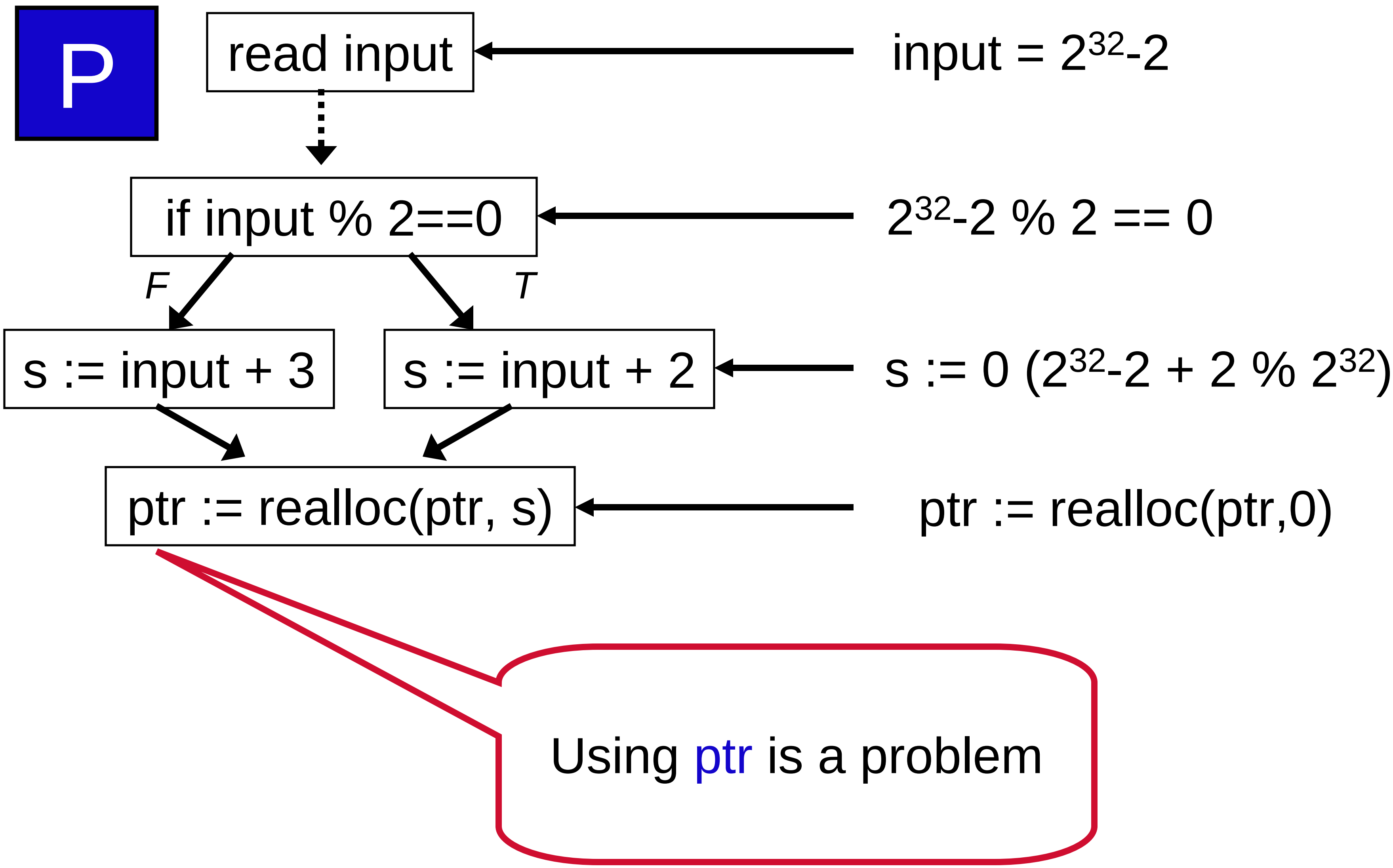
- **Why care?**
  - **Exploits worth money**
    - » **Typically $10,000 - $100,000**
    - » **Great source of research funding :-)**
  - **Know thy enemy**
    - » **Security of patch distribution schemes?**
    - » **Can a patch make you more vulnerable?**
  - **Patch testing**

Running Example

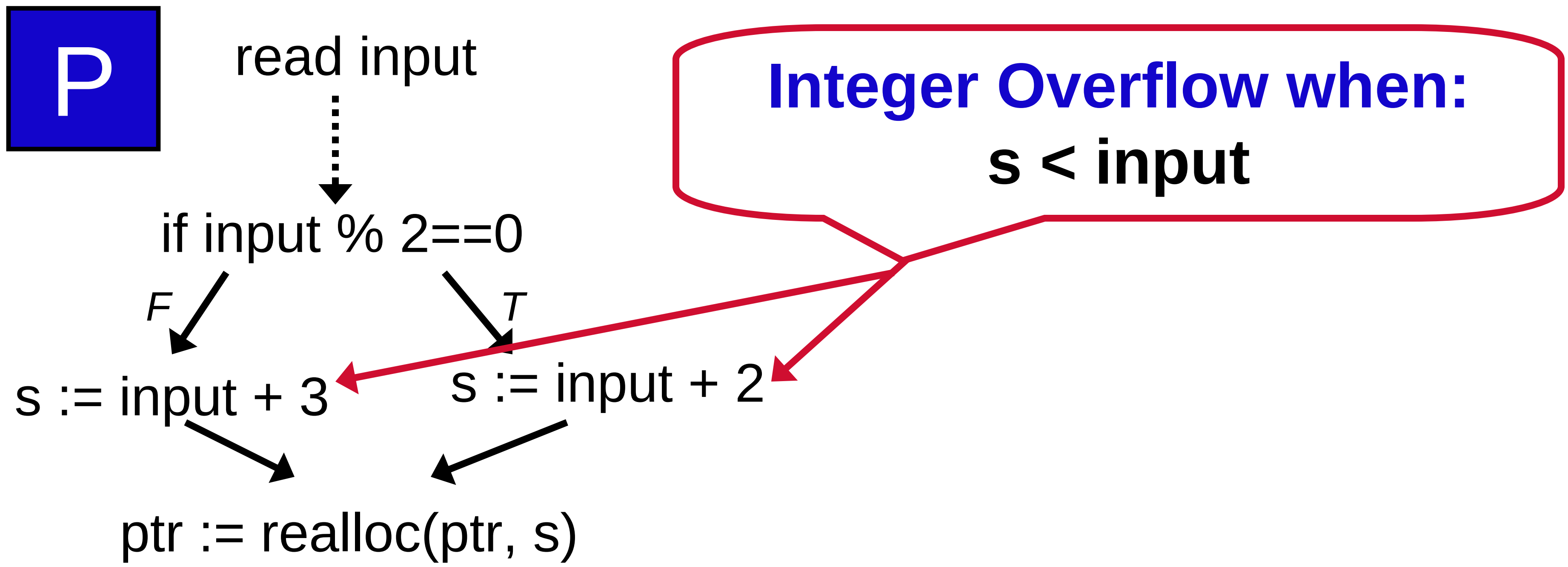


- **All integers unsigned 32-bits**
- **All arithmetic mod 2^{32}**
- **Motivated by real-world vulnerability**

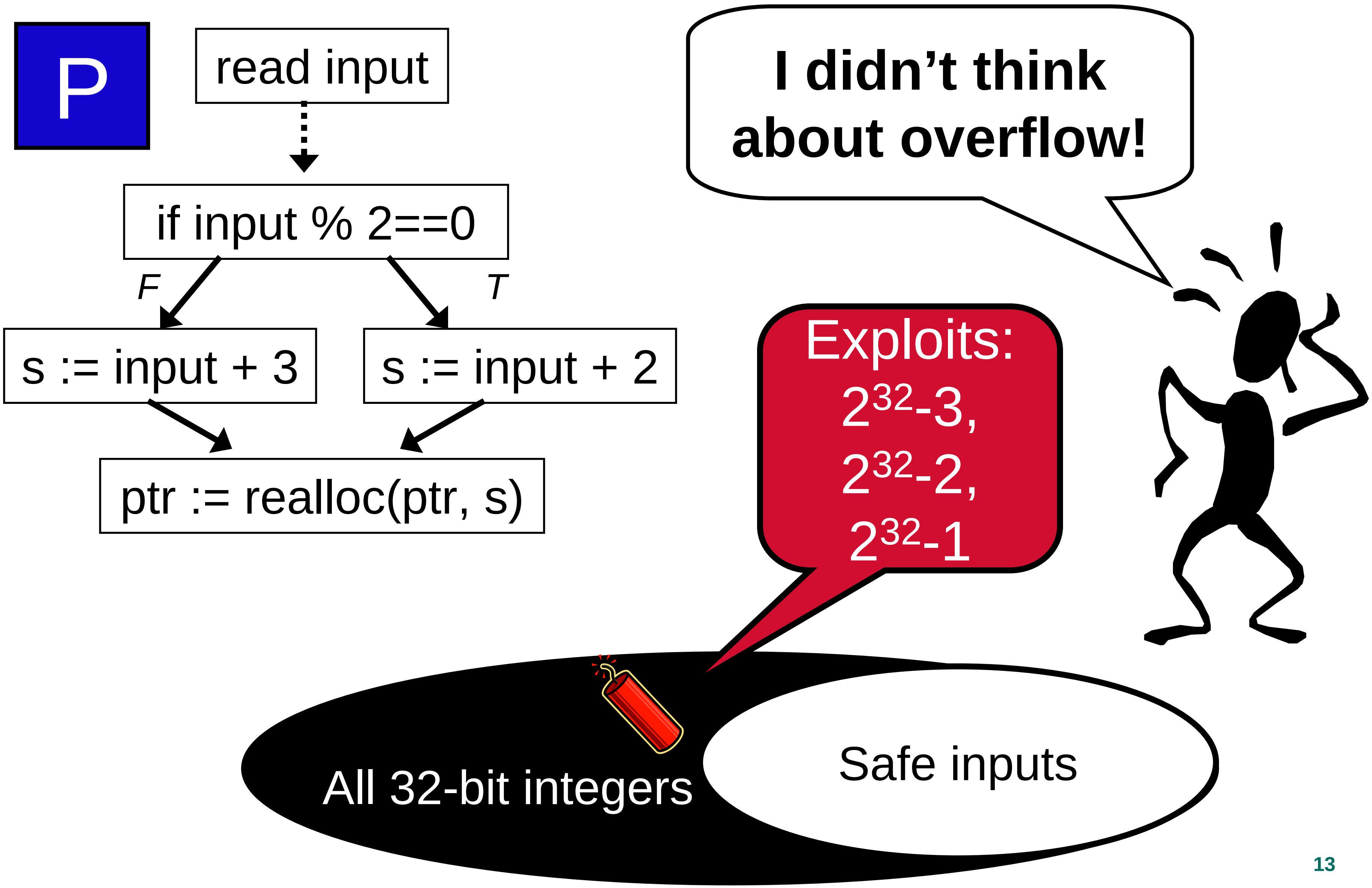
Running Example

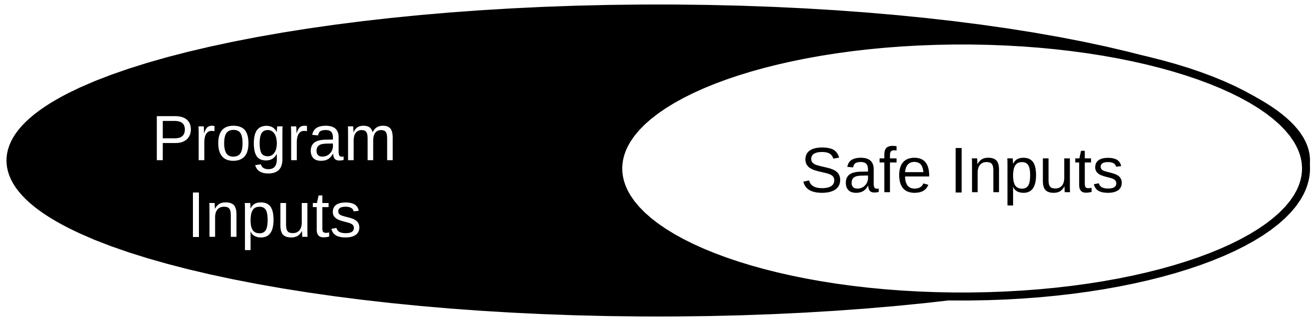


Running Example



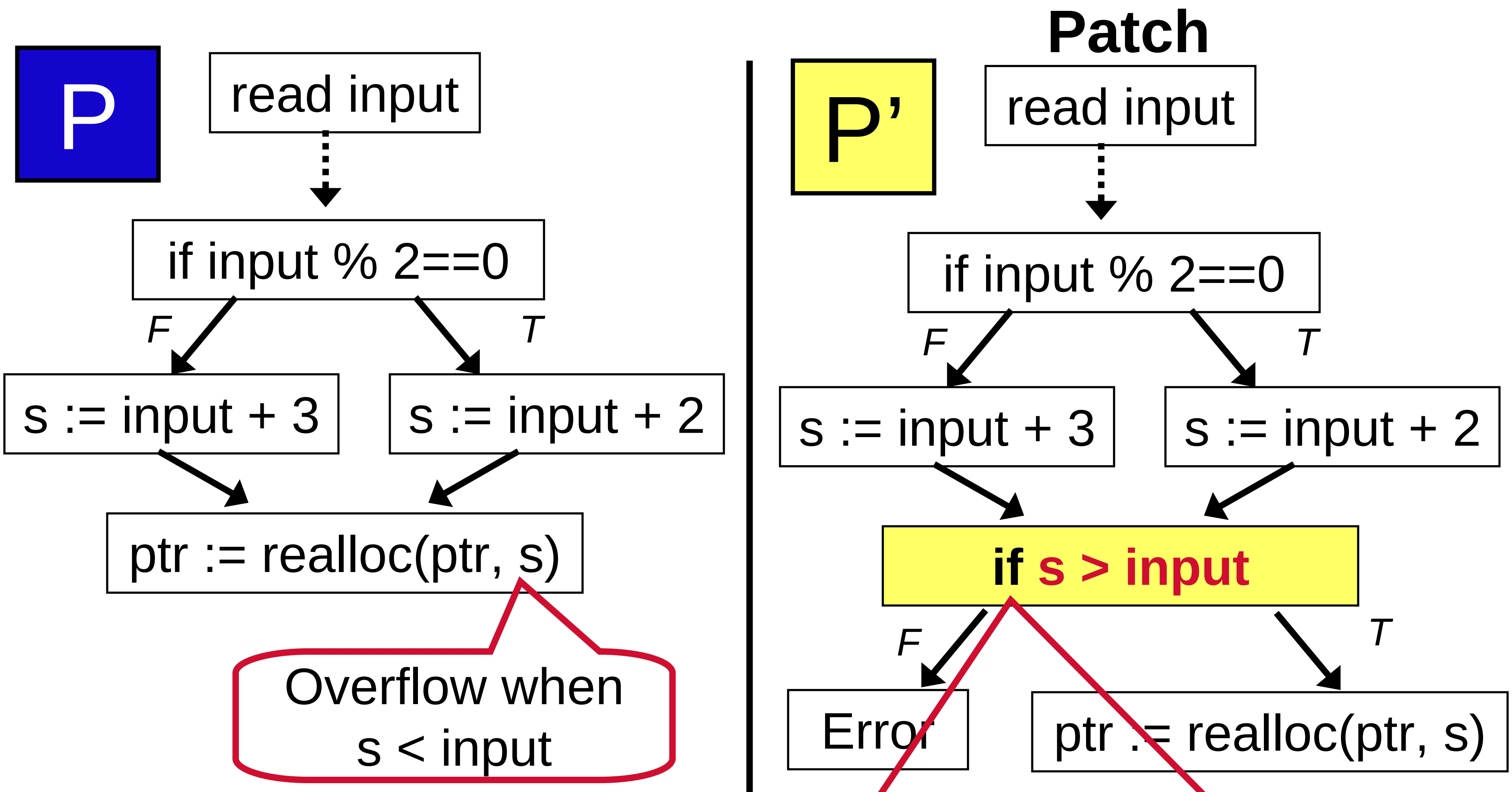
Running Example





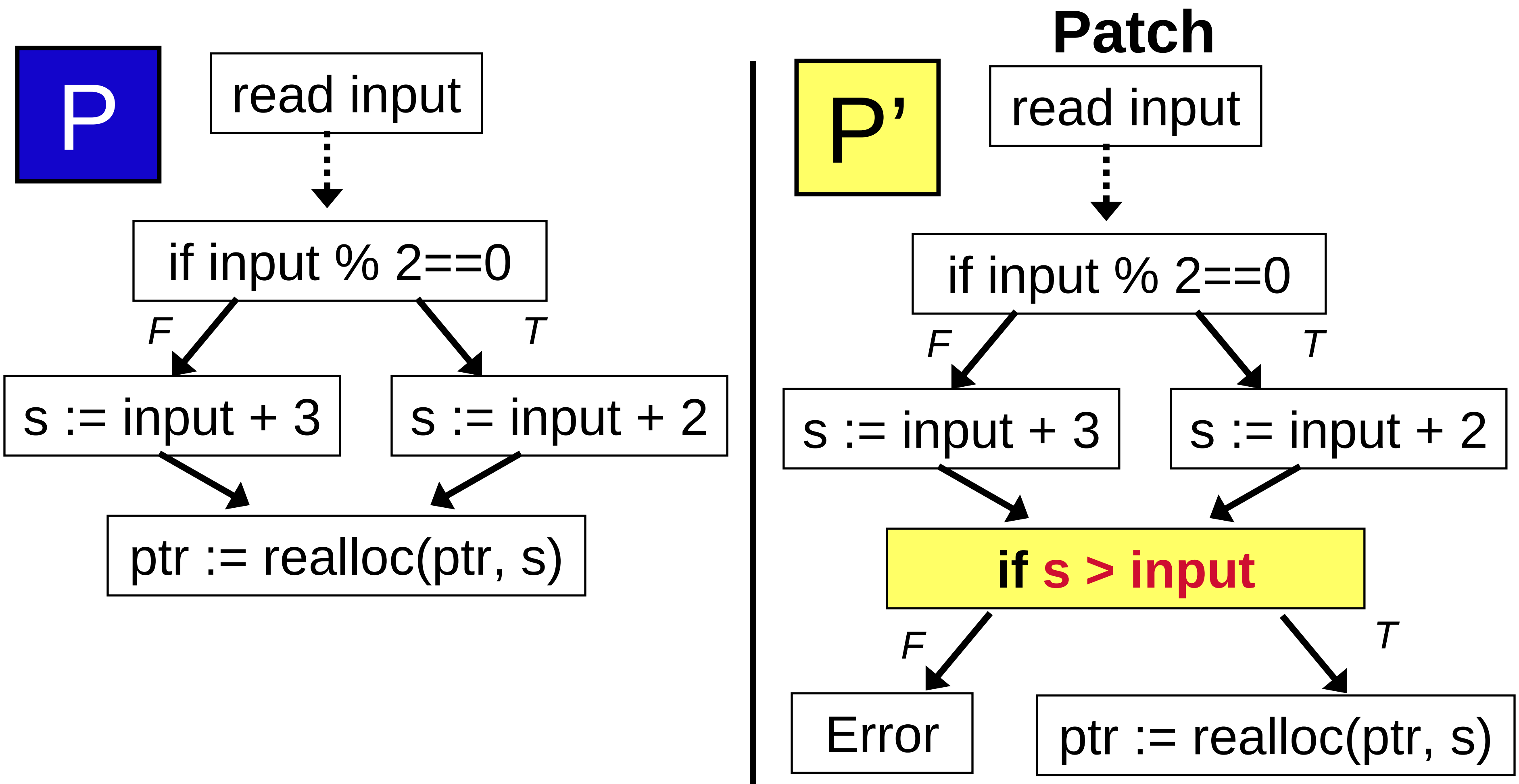
Input Validation Vulnerability

- **Programmer fails to sanitize inputs**
- **Large class of security-critical vulnerabilities**
  - **"Buffer overflow", "integer overflow", "format string vulns", etc.**
- **Responsible for many, many compromised computers**



Patch leaks

1. **Vulnerability point** (where in code)
2. **Vulnerability condition** (under what conditions)

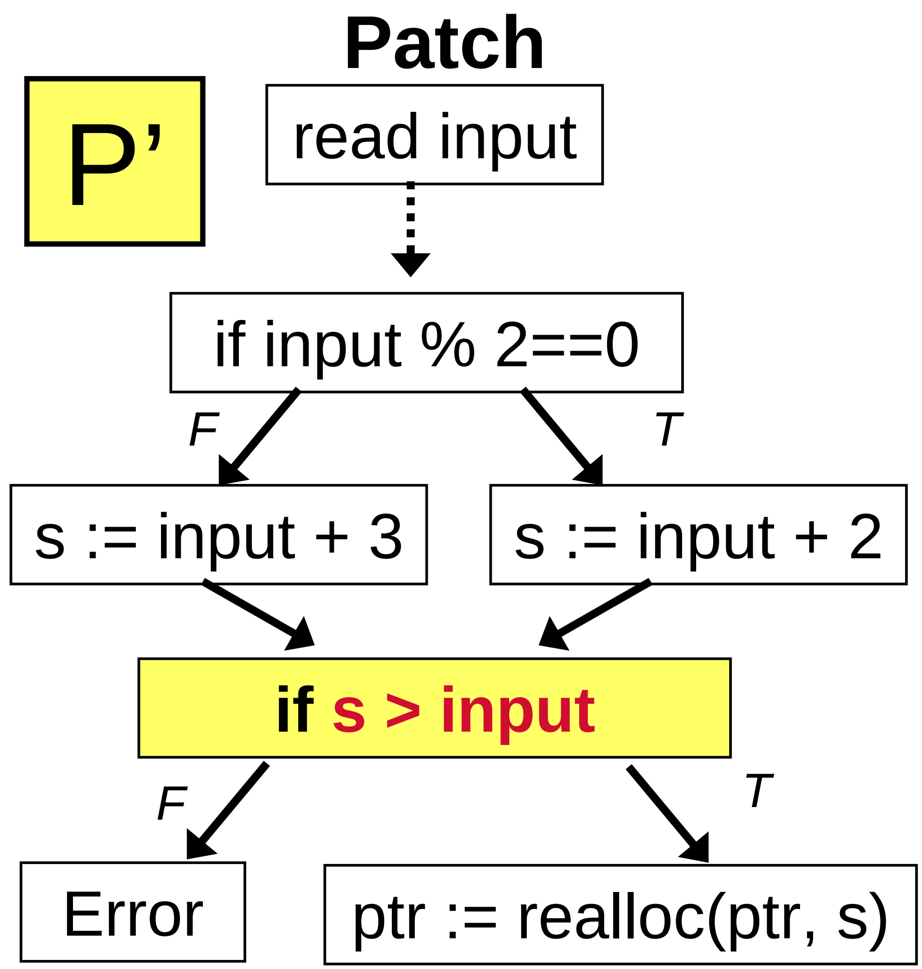


Exploits for P are inputs that fail
vulnerability condition at vulnerability point
(s > input) = false

Our Approach for Patch-based Exploit Generation (I)

Exploit Generation

1. **Diff P and P’ to identify candidate vuln point and condition**
2. **Create input that satisfy candidate vuln condition in P’**
   - **i.e., candidate exploits**
3. **Check candidate exploits on P**



Our Approach for Patch-based Exploit Generation (II)

- **Diff P and P' to identify candidate vuln point and condition**
  - **Currently only consider inserted sanity checks**
  - **Use binary diffing tools to identify inserted checks**
    - » **Existing off-the-shelf syntactic diffing tools**
    - » **BinHunt: our semantic diffing tool**
- **Create candidate exploits**
  - **i.e., input that satisfy candidate vuln condition in P'**
- **Validate candidate exploits on P**
  - **E.g., dynamic taint analysis (TaintCheck)**

Create Candidate Exploits

- **Given candidate vulnerability point & condition**
- **Compute Weakest Precondition over program paths**
  - **Using vulnerability condition as post condition**
  - **Construct formulas representing conditions on input**
    - **Whose execution path included**
    - **Satisfying the vulnerability condition at vulnerability point**
- **Solve formula using solvers**
  - **E.g., decision procedures**
  - **Satisfying answers are candidate exploits**

Different Approaches for Creating Formulas

- **Statically computing formula**
  - **Covering many paths (without explicitly enumerating them)**
  - **Sometimes hard to solve formula**
- **Dynamically computing formula**
  - **Formula easier to solve**
  - **Covering only one path**
- **Combined dynamic and static approach**
  - **Covering multiple paths**
  - **Tune for formula complexity**
- **Experimental results**
  - **Different approach effective for different scenarios**
- **Other techniques to make formulas smaller and easier to solve**

Experimental Results

- **5 Microsoft patches**
  - **Mostly 2007**
  - **Integer overflow, buffer overflow, information disclosure, DoS**
- **Automatically generated exploits for all 5 patches**
  - **In seconds to minutes**
  - **3 out of 5 have no publicly available exploits**
  - **Automatically generated exploit variants for the other 2**
- **Diffing time**
  - **A few minutes**

Exploit Generation Results

Time (s)	DSA_SetItem	ASPNet _Filter	GDI	IGMP	PNG
Dynamic Total	5.68	11.57	10.34	N/A	N/A
Formula	5.51	4.64	10.33	N/A	N/A
Solver	0.17	6.93	0.01	N/A	N/A
Static Total	83.47	N/A	26.41	N/A	N/A
Formula	2.32	N/A	4.99	N/A	N/A
Solver	81.15	N/A	21.42	N/A	N/A
Combined	11.51	N/A	29.07	13.57	104.28
Forumla	6.72	N/A	25.29	13.31	104.14
Solver	4.79	N/A	3.78	0.26	0.14

When could technique fail?

- Decision procedure cannot solve C
- Exploit depends on several conditions in P’ (works in some cases)
- etc.

However, security design must conservatively estimate attackers capabilities

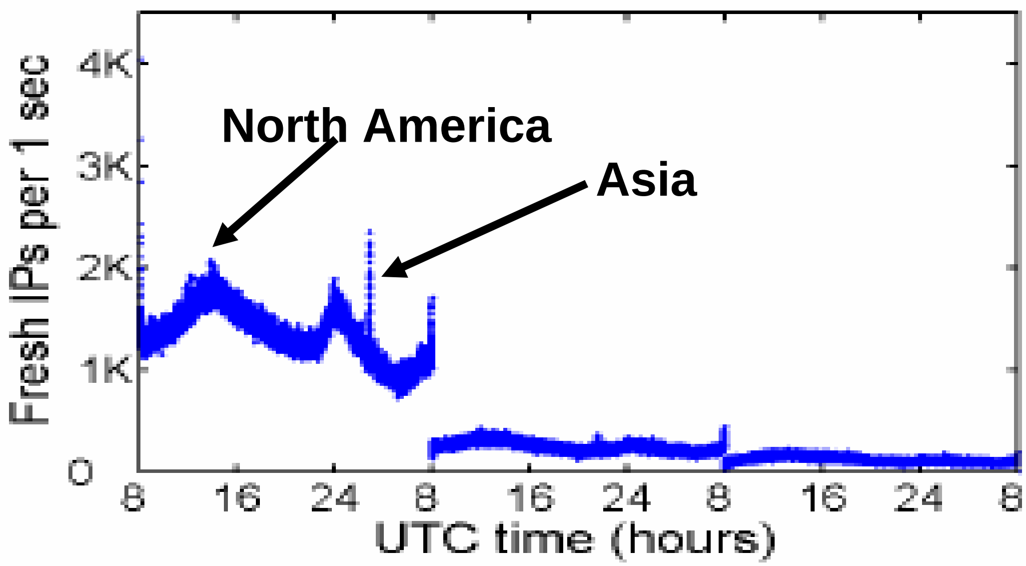
We generate exploits in seconds to minutes

+

Fast worms: ~10 minutes to infect all hosts [2003]

=

Patch release can create serious threats

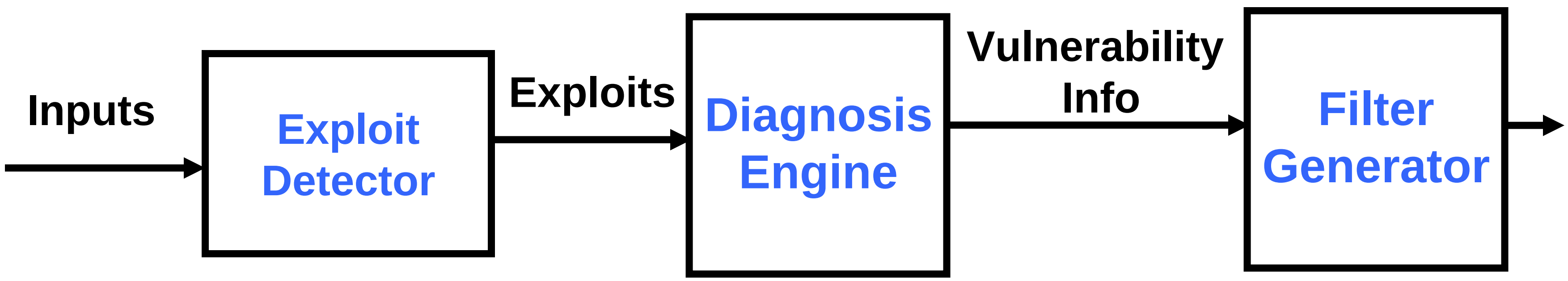


Unique IP's contacting Windows Automatic Update

[GKPV06]

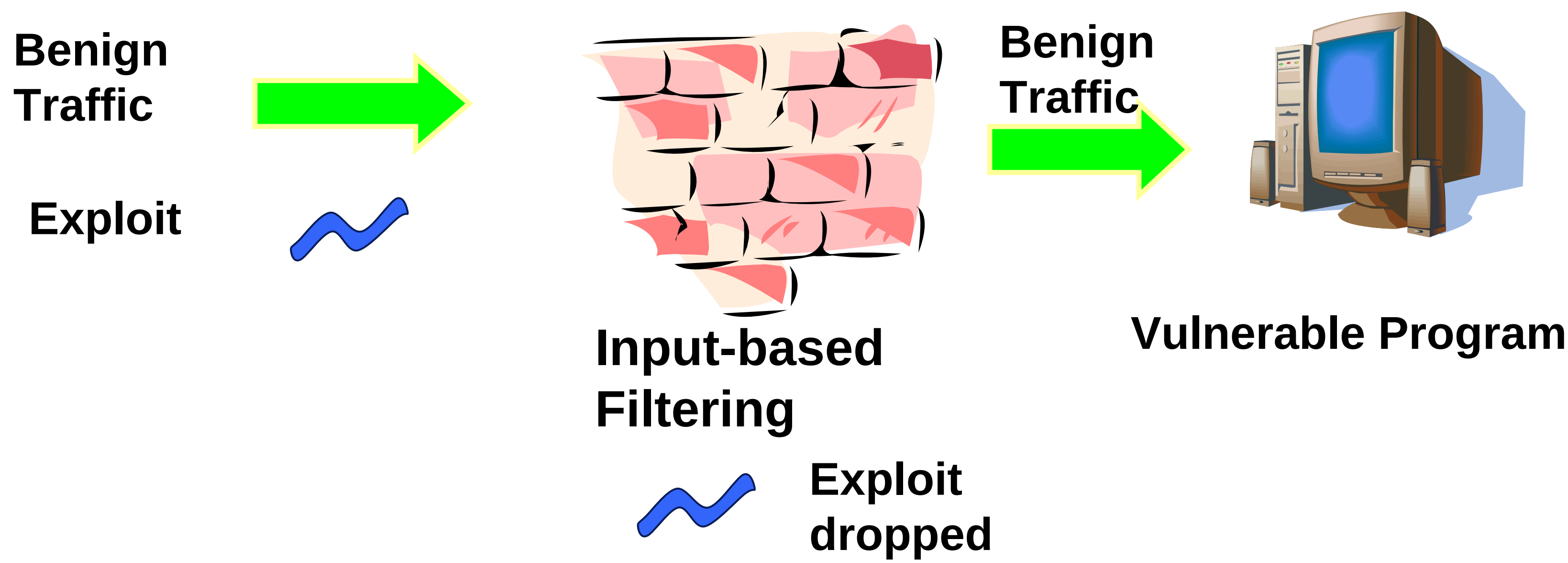
Other Security Applications

- **Effective new approaches for diverse security problems**
  - **Over dozen projects**
  - **Over 12 publications in security conferences**
- **Exploit detection, diagnosis, defense**



- **In-depth malware analysis**
- **Others:**
  - **Reverse engineering**
  - **Deviation detection [Best Paper Award]**
  - **Semantic binary diff**

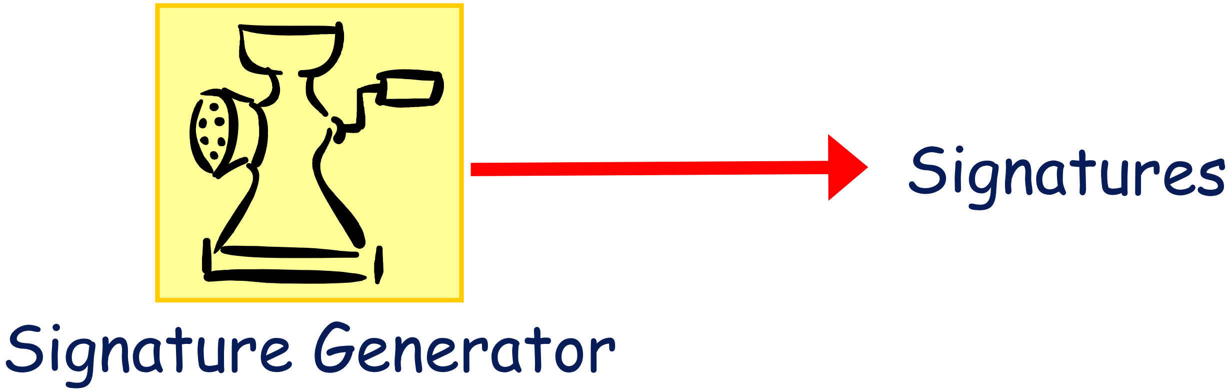
Popular Defense: Input-based Filtering---Block out Exploits



- **Input-based filtering**
  - **Signature f: given input x, f(x) = exploit or benign**
  - **Effective, widely-deployed defense**
- **Central question:**
  How to generate signatures, esp. for new attacks?

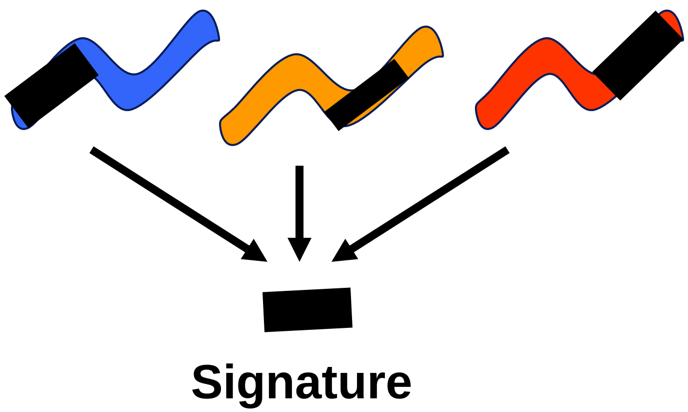
Signature Generation

- **Current common practice: Manual signature generation**
  - **Slow, esp. for zero-day attacks**
  - **Labor-intensive**
  - **Inaccurate**
  - **Limited for scalability & complexity**
- **Our work: automatic generation of vulnerability signatures**



Previous Approaches Insufficient

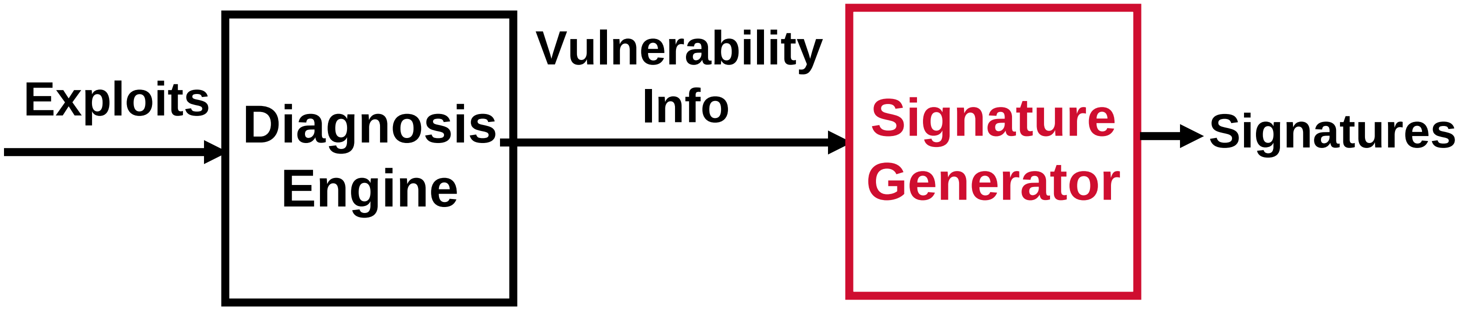
- **Previous approaches: pattern-extraction based**
  - **Extract common patterns in worm samples, not in benign samples**
    - » **Common substring or combination thereof**
    - » **Honeycomb[Kreibich-Hotnets03]**
    - » **Earlybird[Singh-OSDI03]**
    - » **Autograph[Kim-USENIX05]**
    - » **Polygraph[*Song*-IEEE S&P05]**



- **Disadvantages**
  - **Insufficient for polymorphic worms**
    - » **Can't generate signatures for unseen variants**
  - **No guarantee of signature quality**
  - **Susceptible to adversarial learning [*Song*-RAID06]**
  - **Purely bit-pattern syntactic approach, so no semantic understanding of vulnerability**

Automatic Generation of Vulnerability Signatures

- **Instead of bit patterns, use root cause**
  - **Generating signatures based on vulnerability**
- **Given an exploit, first identify vulnerability information**
  - **Vulnerability Point: where the vulnerability is**
  - **Vulnerability Condition: what triggers the vulnerability**
    - » **E.g., condition for buffer overflow**
  - **Using a combination of static & dynamic analysis**
- **Then generate signatures with given vulnerability information**

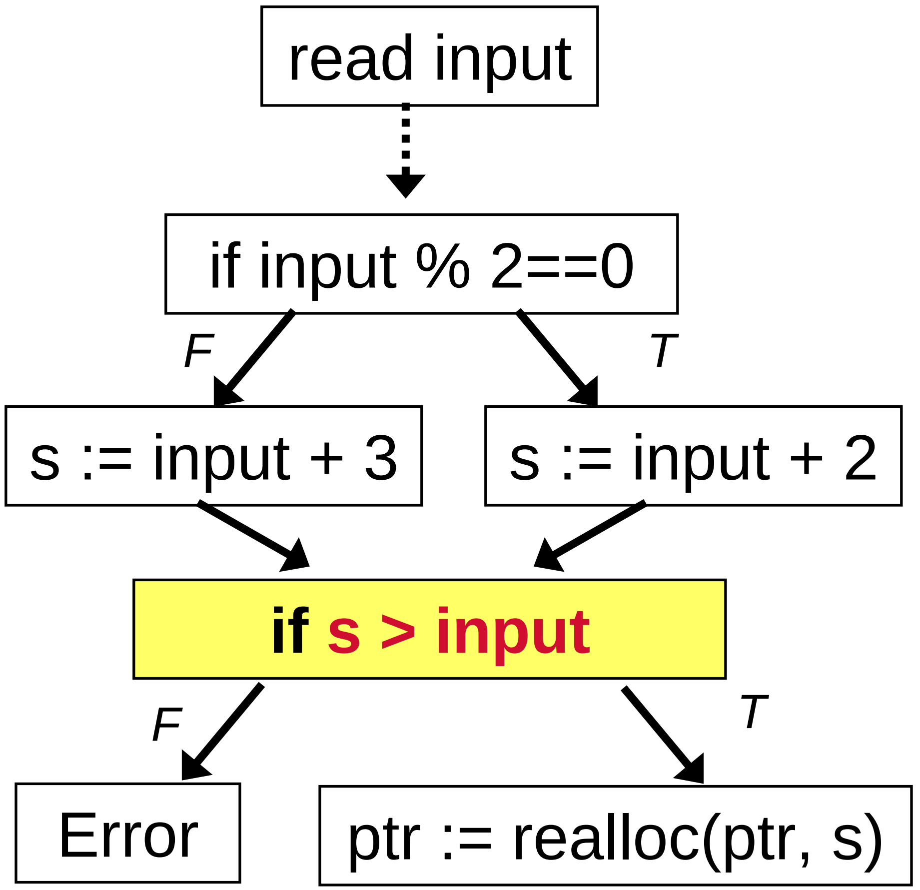


Approach: Extracting Constraints Imposed by Vulnerability

- **As exploits morph, they need to trigger vulnerability**
- **So, vulnerability puts constraints on exploits**
- **Problem reduction:**
  - **Signature generation =**
    constraints on inputs that trigger vulnerability
- **Symbolic execution**
- **Soundness guaranteed (no false positives)**

Automatic Vulnerability Signature Generation

What should the signature be?



Protocol-aware Signatures

- **So far, symbolic constraint signatures operate on bits**
- **Given protocol parsing information (e.g., a parse tree),**
  - **lift constraints to field-level**
  - **Remove parsing related constraints**
  - **Generate symbolic constraint signatures on field-level**
- **Effective for variable-length fields, iterative fields, etc.**
- **Used in conjunction with signature matching engine with protocol parsing capability**

Evaluation: Protocol-aware Signatures

- **Automatically generated optimal or close to optimal signatures for real-world exploits**
  - **SQL, GHttpd, AtpHttpd, GDI, Windows DCOM RPC vulnerabilities**

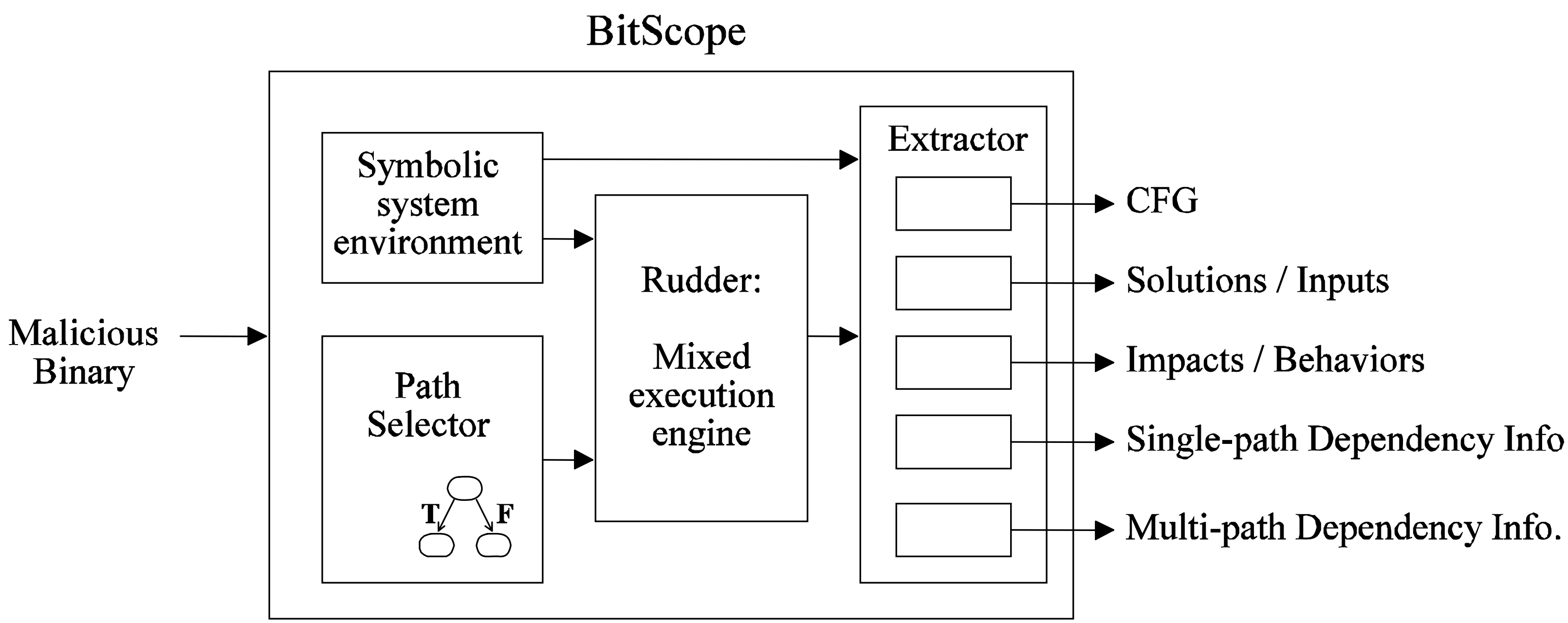
- **Signature for SQL:**
  - **(FIELD_CMD==4) ^ length(FIELD_DB) > 64**
- **Signature for GHttpd:**
  - **(strcmp(METHOD, “GET”) != 0 ^ length(METHOD) > 165) ||**
    (strcmp(METHOD, “GET) == 0 ^ strstr(URI, “/..”) !=0 ^
    length(URI)>170) ||
    (strcmp(METHOD, “GET”) == 0 ^ strstr(URI, “/..”) == 0 ^
    length(URI) + length(ClientAddr) > 166)

In-depth Malware Analysis

- **High volume of new malware needs automatic malware analysis**
- **Given a piece of suspicious code sample,**
  - **What malicious behaviors will it have?**
  - **How to classify it?**
    - **Key logger, BHO Spyware, Backdoor, Rootkit**
  - **What mechanisms does it use?**
    - **How does it steal information?**
    - **How does it hook?**
  - **Who does it communicate with? Where does it send information to?**
  - **Does its communication exhibit certain patterns?**
  - **Does it contain trigger-based behavior?**
    - **Time bombs**
    - **Botnet commands**
- **Can we design & develop a unified framework to answer these questions?**

BitScope: THE In-depth Malware Analysis infrastructure

- **Identify/analyze malicious behavior based on root cause**
  - **Privacy-breaching malware: spyware, keylogger, backdoor, etc.**
  - **Malware perturbing system by hooking: rootkit, etc.**
- **Understand how malware get into the system**
  - **What mechanisms/vulnerabilities does it exploit**
- **Explore hidden behavior, detect trigger-based behavior**
  - **Automatically identifying botnet program commands, time bombs**



BitBlaze Malware Analysis Online

- **A subset of our malware analysis functionalities**
  - **Malware unpacking, IDA-Pro plug-in**
  - **Extracting behaviors**
- **Parallel architecture for high-volume malware analysis**
- **Public service:**
  - **Submit malware samples and get results back**

Outline

- **BitBlaze in action: sample security applications**
  - **Automatic patch-based exploit generation**
  - **In-depth malware analysis and other applications**
- **BitBlaze Binary Analysis Infrastructure**
  - **Challenges**
  - **Design rationale**
  - **Architecture**
- **Future directions of binary analysis & beyond**

BitBlaze Binary Analysis Infrastructure: Challenges

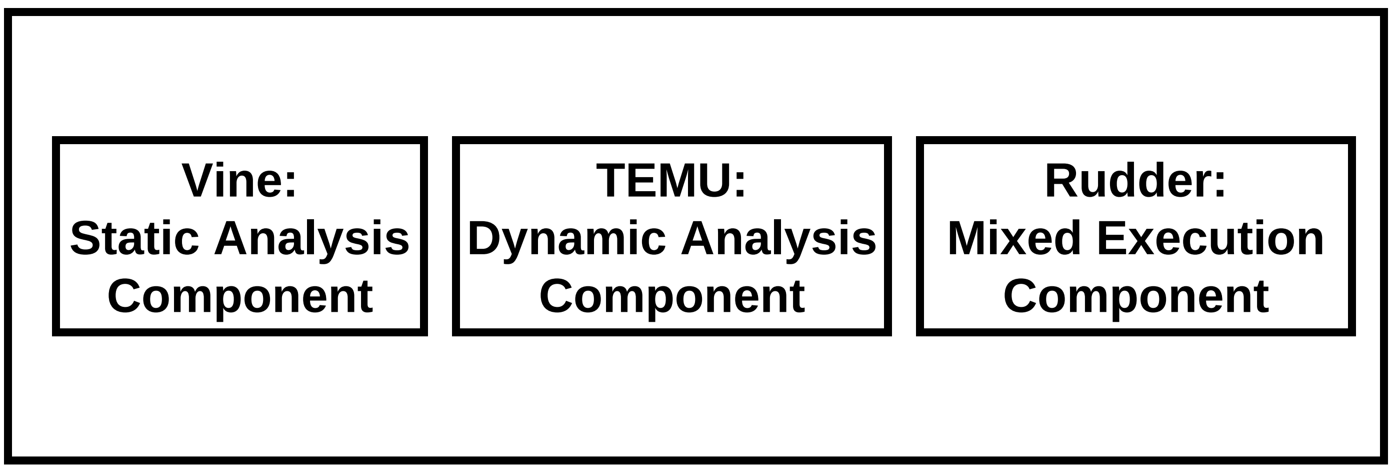
- **Complexity**
  - **IA-32 manuals for x86 instruction set weights over 11 pounds**
- **Lack higher-level semantics**
  - **Even disassembling is non-trivial**
- **Require whole-system view**
  - **Operations within kernel and interactions btw processes**
- **Malicious code may obfuscate**
  - **Code packing**
  - **Code encryption**
  - **Code obfuscation & dynamically generated code**

BitBlaze Binary Analysis Infrastructure: Design Rationale

- **Accuracy**
  - **Enable precise analysis, formally modeling instruction semantics**
- **Extensibility**
  - **Develop core utilities to support different architecture and applications**
- **Fusion of static & dynamic analysis**
  - **Static analysis**
    - **Pros: more complete results**
    - **Cons: pointer aliasing, indirect jumps, code obfuscation, kernel & floating point instructions difficult to model**
  - **Dynamic analysis**
    - **Pros: easier**
    - **Cons: limited coverage**
  - **Solution: combining both**

BitBlaze Binary Analysis Infrastructure: Architecture

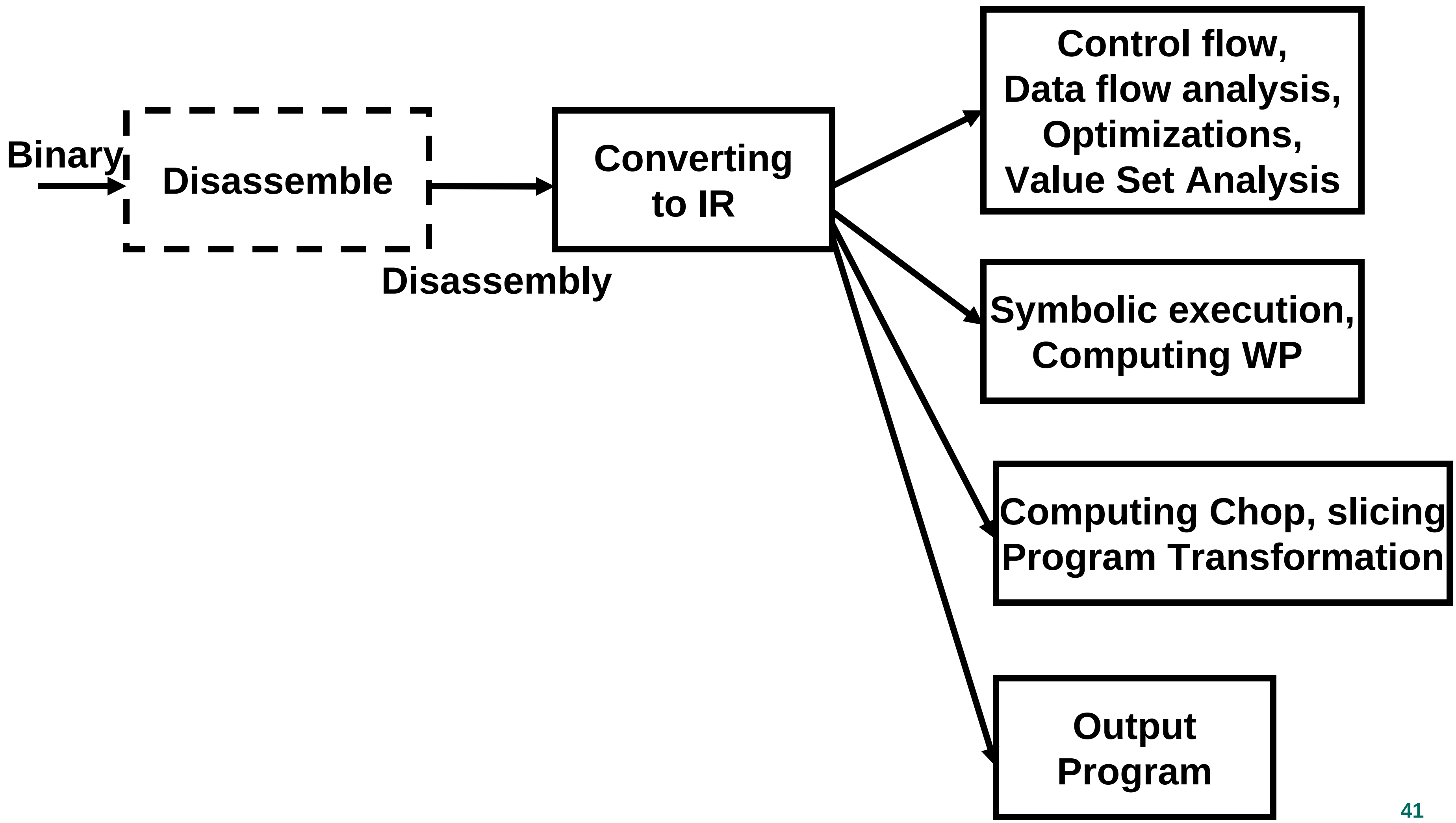
- **The first infrastructure:**
  - **Novel fusion of static, dynamic analysis techniques, and formal analysis techniques such as symbolic execution & abstract interpretation**
  - **Capable of analyzing whole system (including OS kernel)**
  - **Capable of analyzing packed/encrypted/obfuscated code**



BitBlaze Binary Analysis Platform

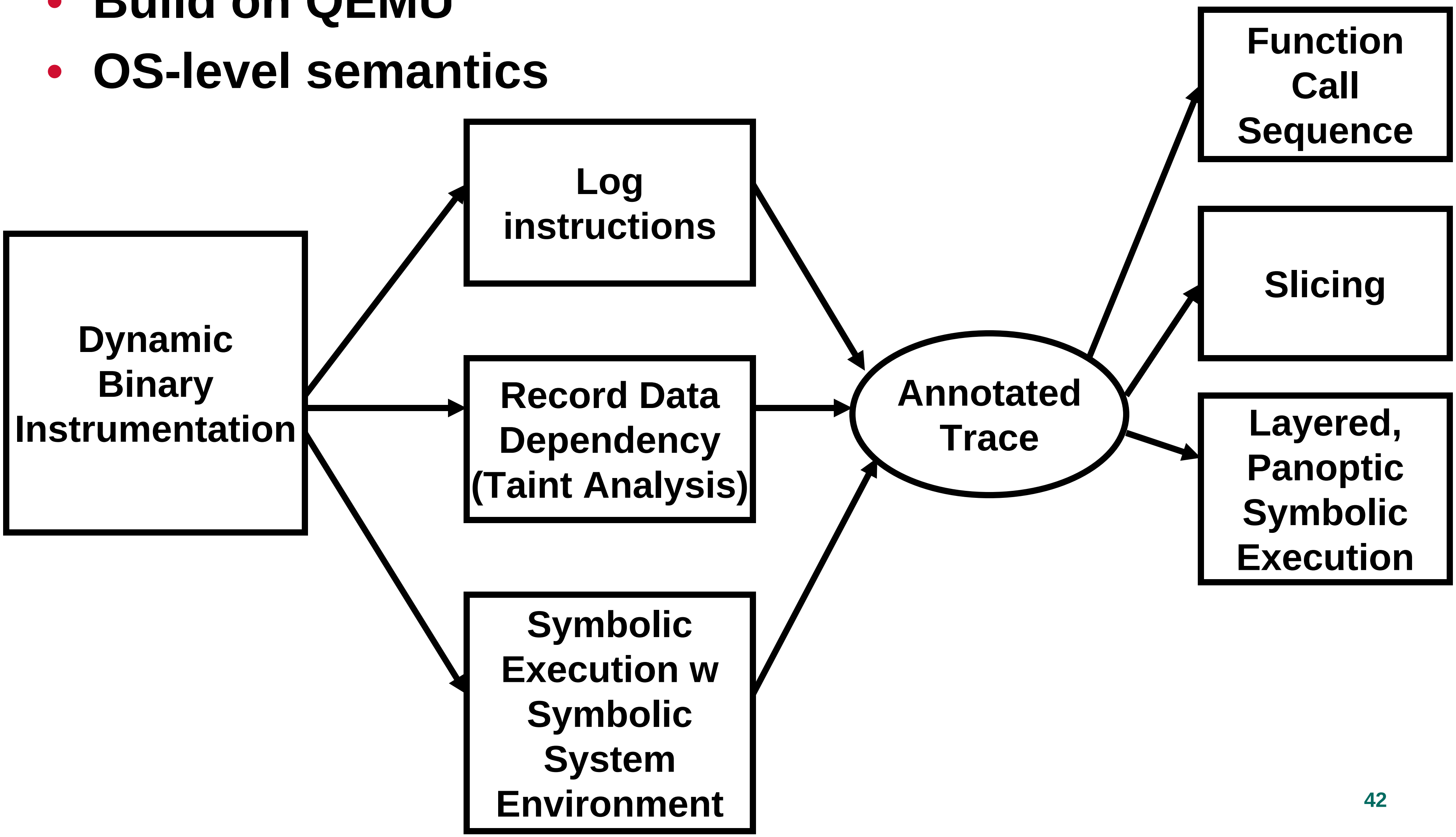
Vine

- **Static analysis component**



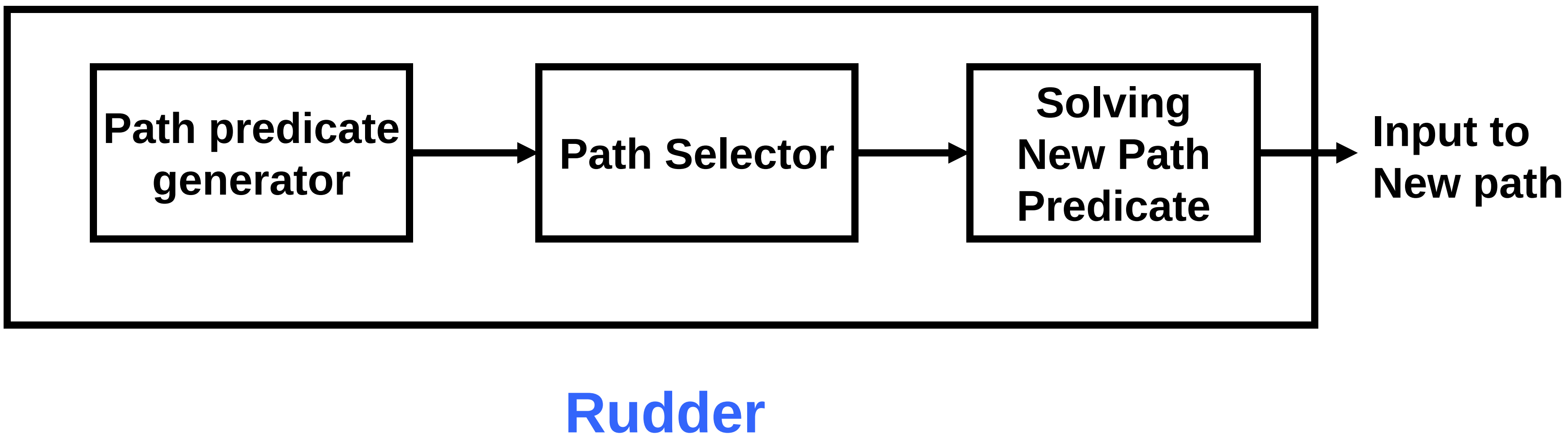
TEMU

- Work for both Windows & Linux, applications & kernel
- Build on QEMU
- OS-level semantics



Rudder

- Compute path predicate
- Obtain new path predicate by reverting branches
- Solve path predicate to obtain new input to go down a different path



Outline

- **BitBlaze Binary Analysis Infrastructure**
  - **Challenges**
  - **Design rationale**
  - **Architecture**

- **BitBlaze in action: sample security applications**
  - **Automatic patch-based exploit generation**
  - **In-depth malware analysis**

- **Future directions of binary analysis & beyond**

The Vision

- **Binary-only code audit and assurance**
  - **Given a third-party program**
  - **Does it have vulnerabilities?**
  - **Does it have certain security guarantees?**
  - **Does it contain trojans?**

- **Design and develop an infrastructure to accomplish this**
  - **More advanced binary analysis and program verification techniques**
  - **Annotated binaries**
  - **Holistic solution including the software development cycle**

Conclusion

- **BitBlaze binary analysis platform**
  - **A unique fusion of dynamic, static analysis & formal analysis**
- **Solutions to broad spectrum of security applications**
  - **Vulnerability discovery, diagnosis, defense**
  - **In-depth malware analysis**
  - **Automatic model extraction and analysis**
- **Important future research direction**

Contact

- http://bitblaze.cs.berkeley.edu

- dawnsong@cs.berkeley.edu

- **BitBlaze team:**
  David Brumley, Juan Caballero, Ivan Jager, Cody Hartwig, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, Prateek Saxena, Heng Yin