BitBlaze: Binary Analysis for Computer Security

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**

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**
   - **Other security applications: network decoding, etc.**

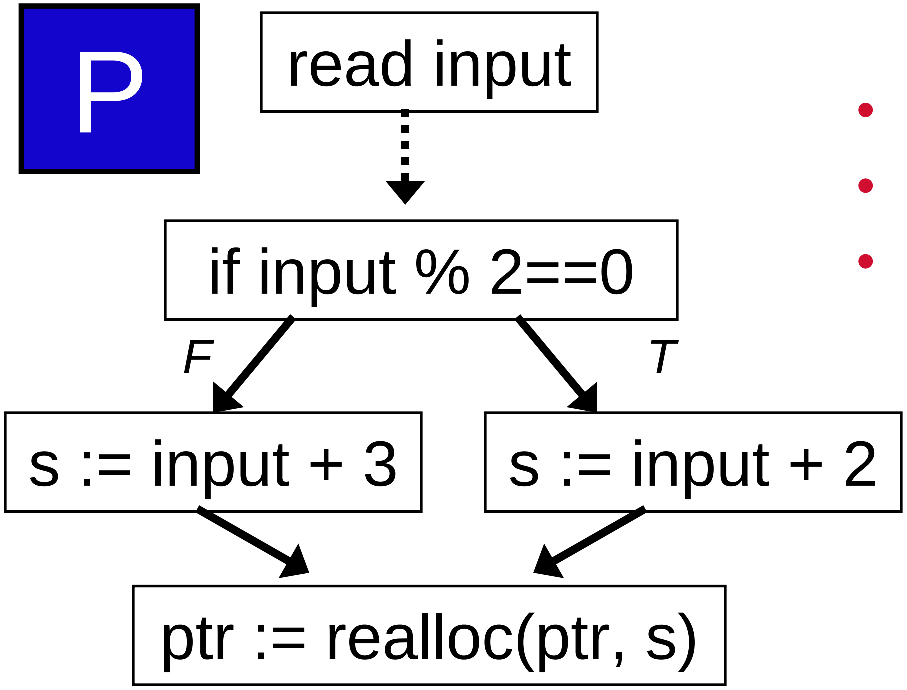
Patch Tuesday

- **Given a patch release, how can we protect vulnerable hosts & networks who have not patched yet?**

- **#1: patch release creates serious threat**
  - **Automatic patch-based exploit generation can generate exploits in minutes**

- **#2: automatic vulnerability signature generation provides instant protection**

Running Example



P

read input

if input % 2==0

F

T

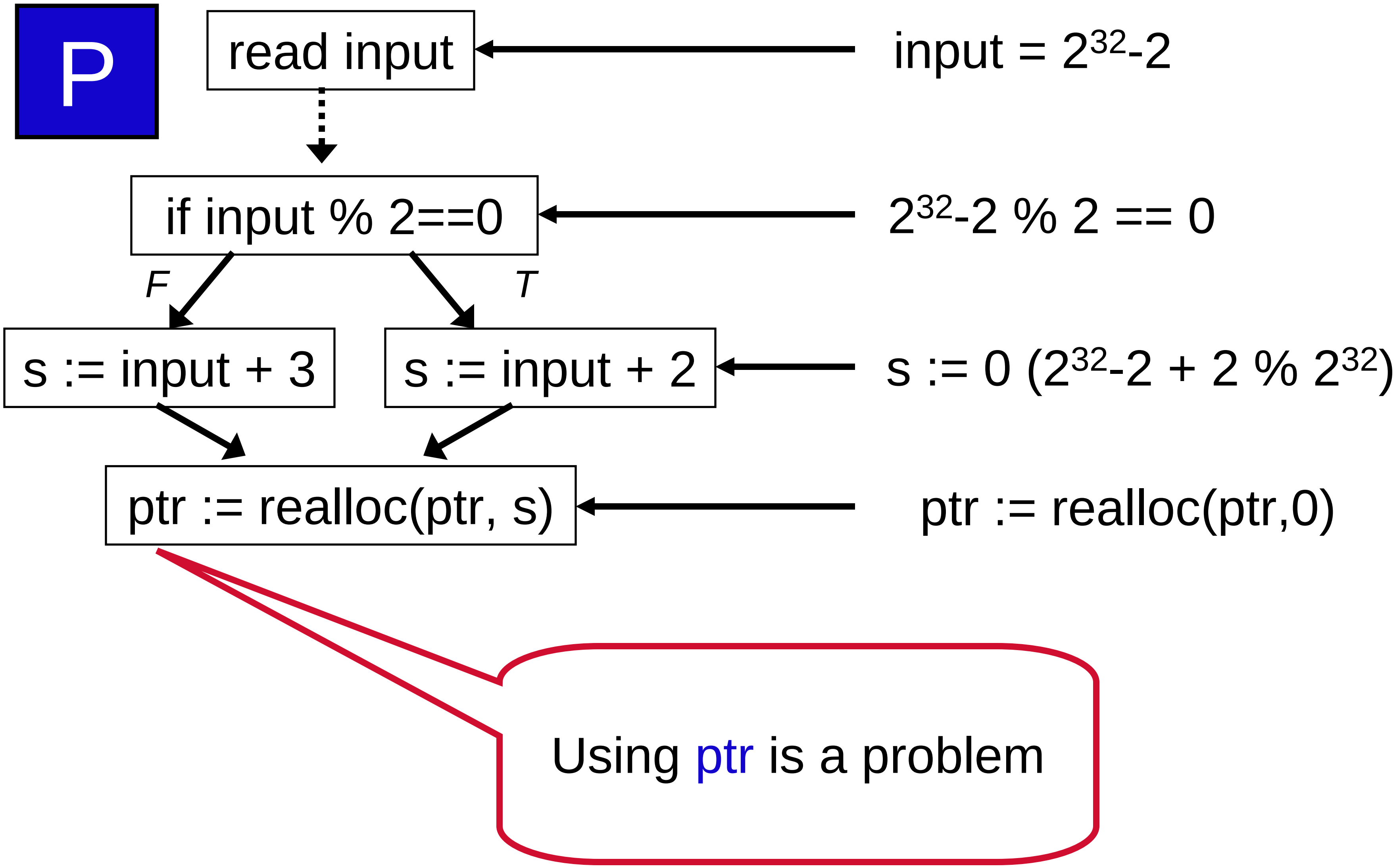
s := input + 3

s := input + 2

ptr := realloc(ptr, s)

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

Running Example



P

read input

input = 2^{32}-2

if input % 2==0

2^{32}-2 % 2 == 0

F

T

s := input + 3

s := input + 2

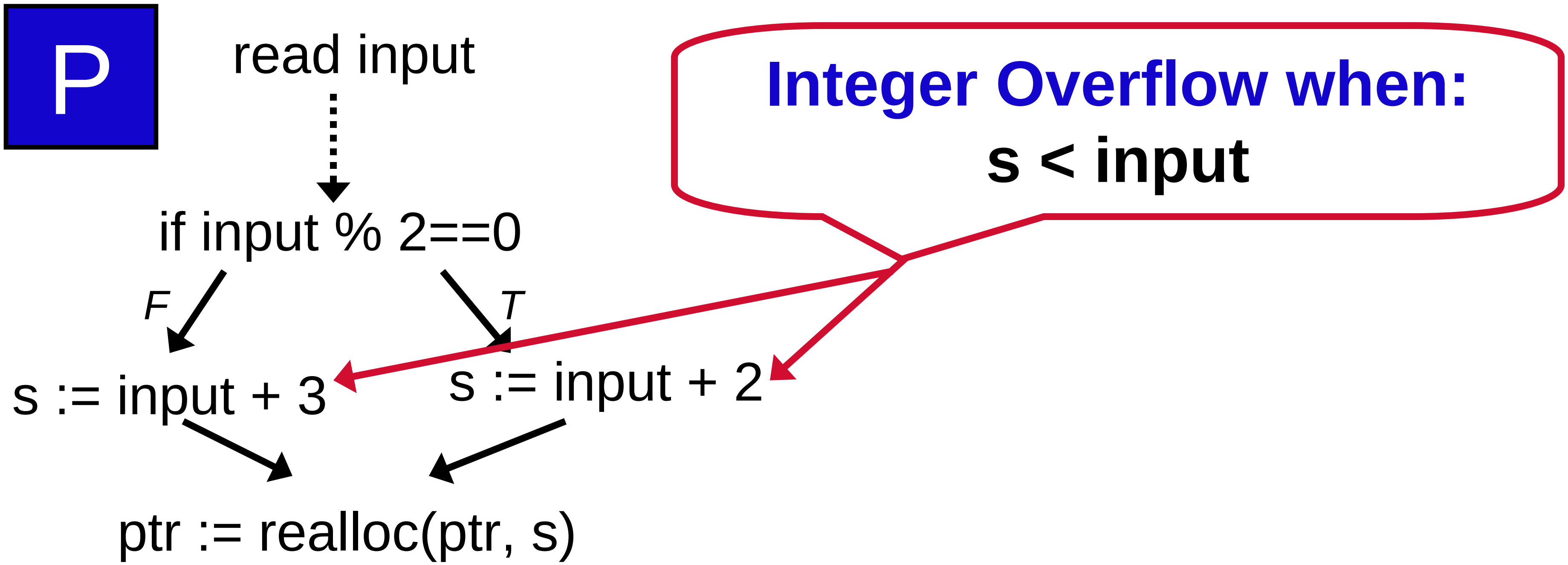
s := 0 (2^{32}-2 + 2 % 2^{32})

ptr := realloc(ptr, s)

ptr := realloc(ptr,0)

Using ptr is a problem

Running Example



P

read input

if input % 2==0

F

T

s := input + 3

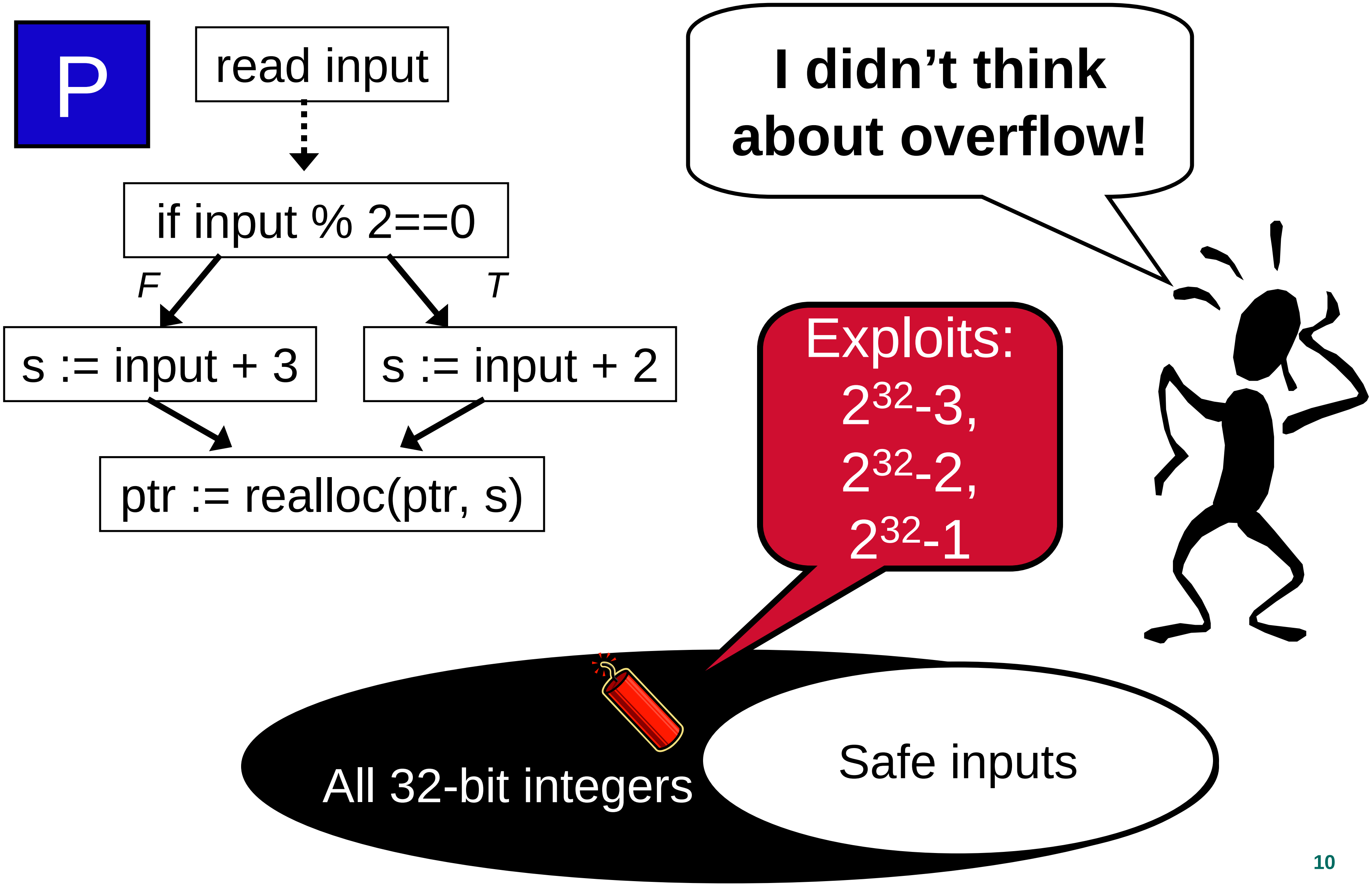
s := input + 2

ptr := realloc(ptr, s)

Integer Overflow when:

s < input

Running Example



P

read input

if input % 2==0

F

T

s := input + 3

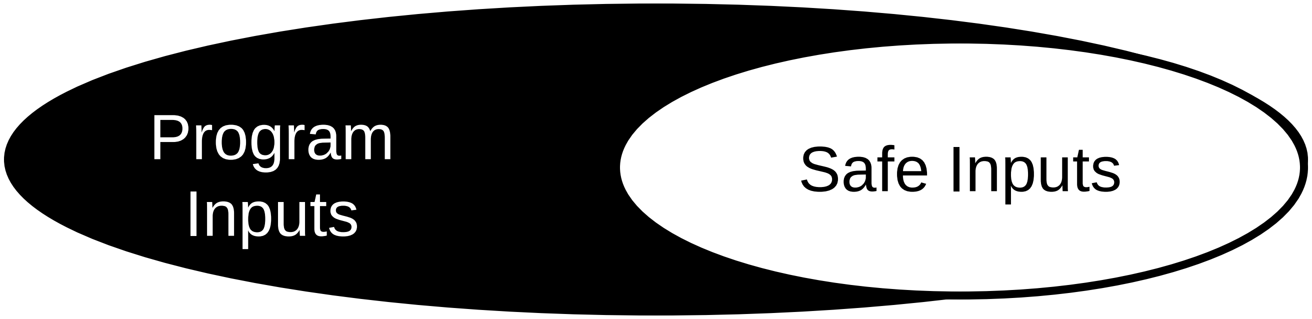
s := input + 2

ptr := realloc(ptr, s)

Exploits: 2^{32}-3, 2^{32}-2, 2^{32}-1

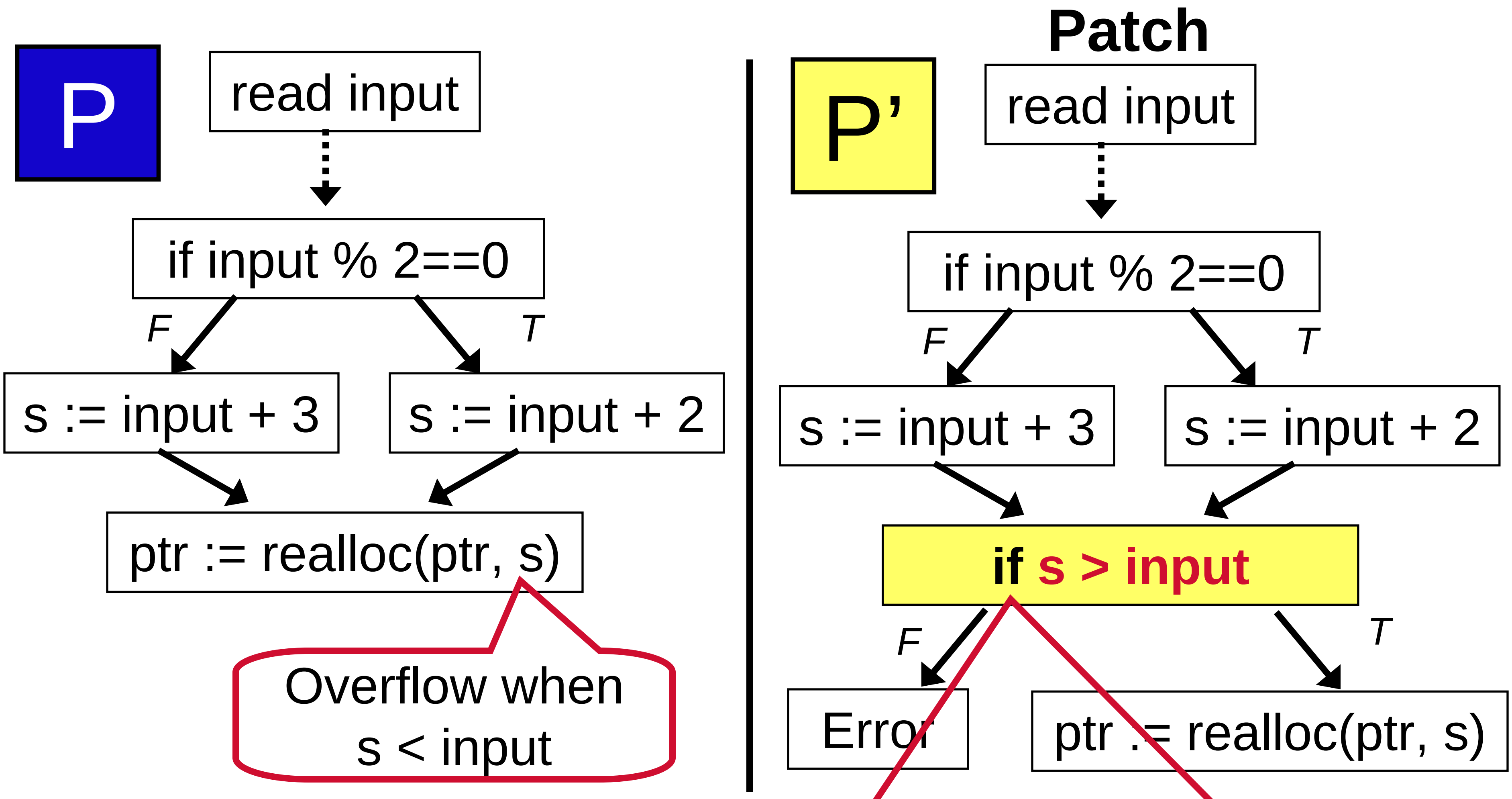
All 32-bit integers

Safe inputs



Input Validation Vulnerability

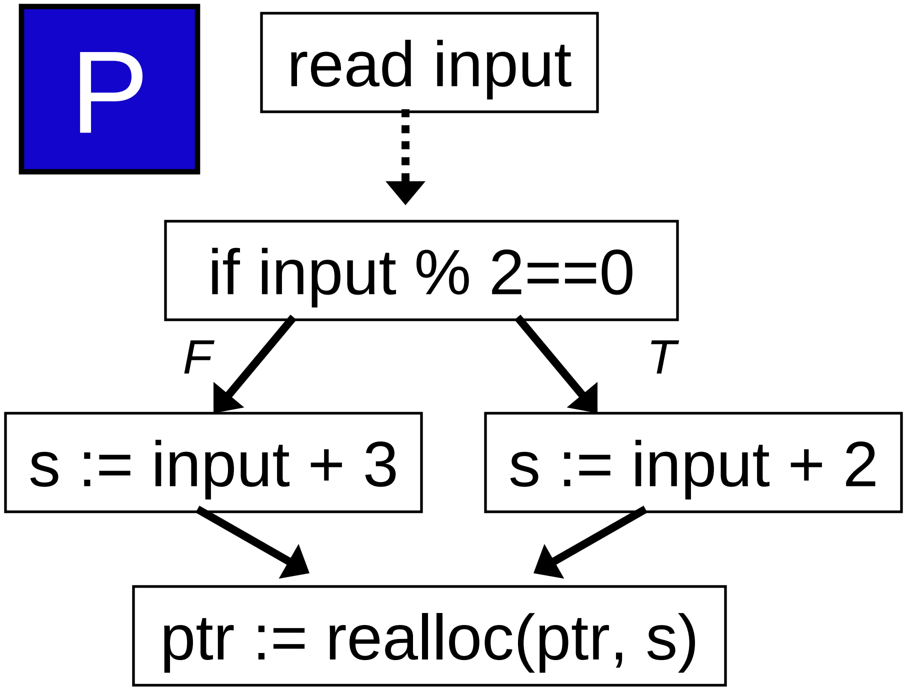
- **Programmer fails to sanitize inputs**
- **Large class of security-critical vulnerabilities**
  - **“Buffer overflow”, “integer overflow”, “format string vulns”, etc.**
- **Responsible for many, many compromised computers**



Overflow when
s < input

Patch leaks

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



P

read input

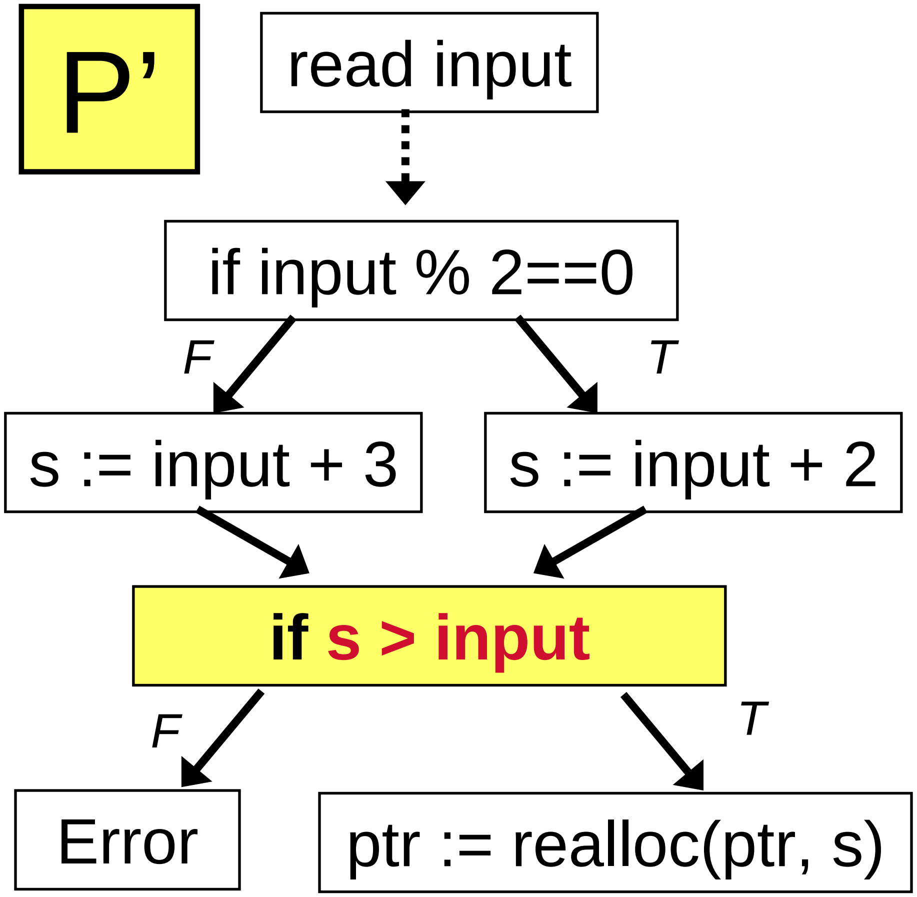
if input % 2==0

s := input + 3

s := input + 2

ptr := realloc(ptr, s)

Patch



P’

read input

if input % 2==0

s := input + 3

s := input + 2

if s > input

Error

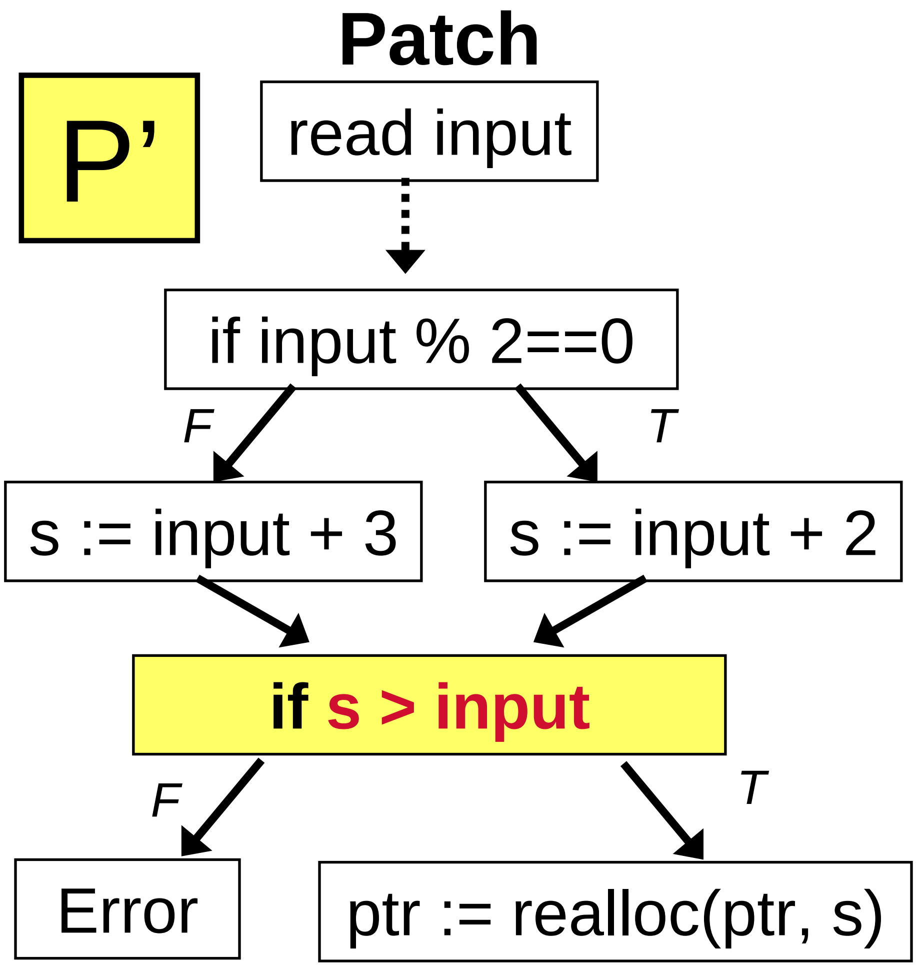
ptr := realloc(ptr, s)

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



P'

Patch

read input

if input % 2==0

F

T

s := input + 3

s := input + 2

if s > input

F

T

Error

ptr := realloc(ptr, s)

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

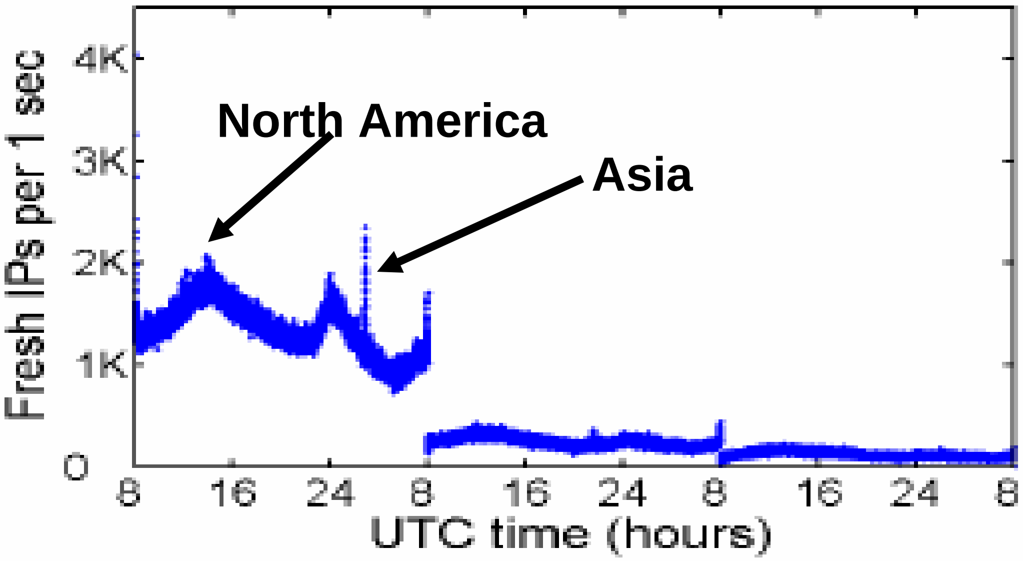
We generate exploits in seconds to minutes

+

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

=

Patch release can create serious threats

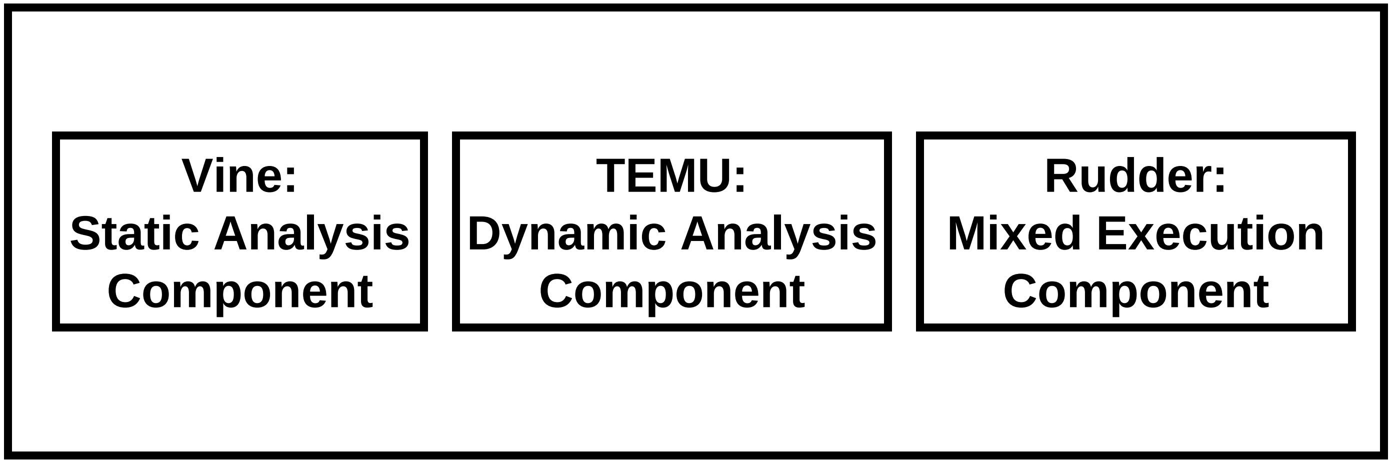


Unique IP’s contacting Windows Automatic Update

[GKPV06]

BitBlaze Binary Analysis Platform

- **Binary analysis infrastructure for security applications:**
  - **Novel fusion of static, dynamic analysis techniques, and program verification techniques**
  - **Whole system analysis (including OS kernel)**
  - **Handling packed/encrypted/obfuscated code**


Vine:
Static Analysis Component
TEMU:
Dynamic Analysis Component
Rudder:
Mixed Execution Component

BitBlaze Binary Analysis Platform

Conclusion

- **Automatically extracting security related properties from binary programs using binary analysis provide a sound approach to many security problems**
  - **Vulnerability diagnosis and signature generation**
  - **In-depth malware analysis**
  - **Deviation Detection**
  - **Network protocol reverse engineering**
- **BitBlaze**
  - **Identify common needs across different security applications**
  - **Fusion of dynamic, static, formal analysis techniques to provide a unified framework for binary analysis for security applications**
- **http://bitblaze.cs.berkeley.edu**

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