BitBlaze: Binary Analysis for Computer Security

Dawn Song

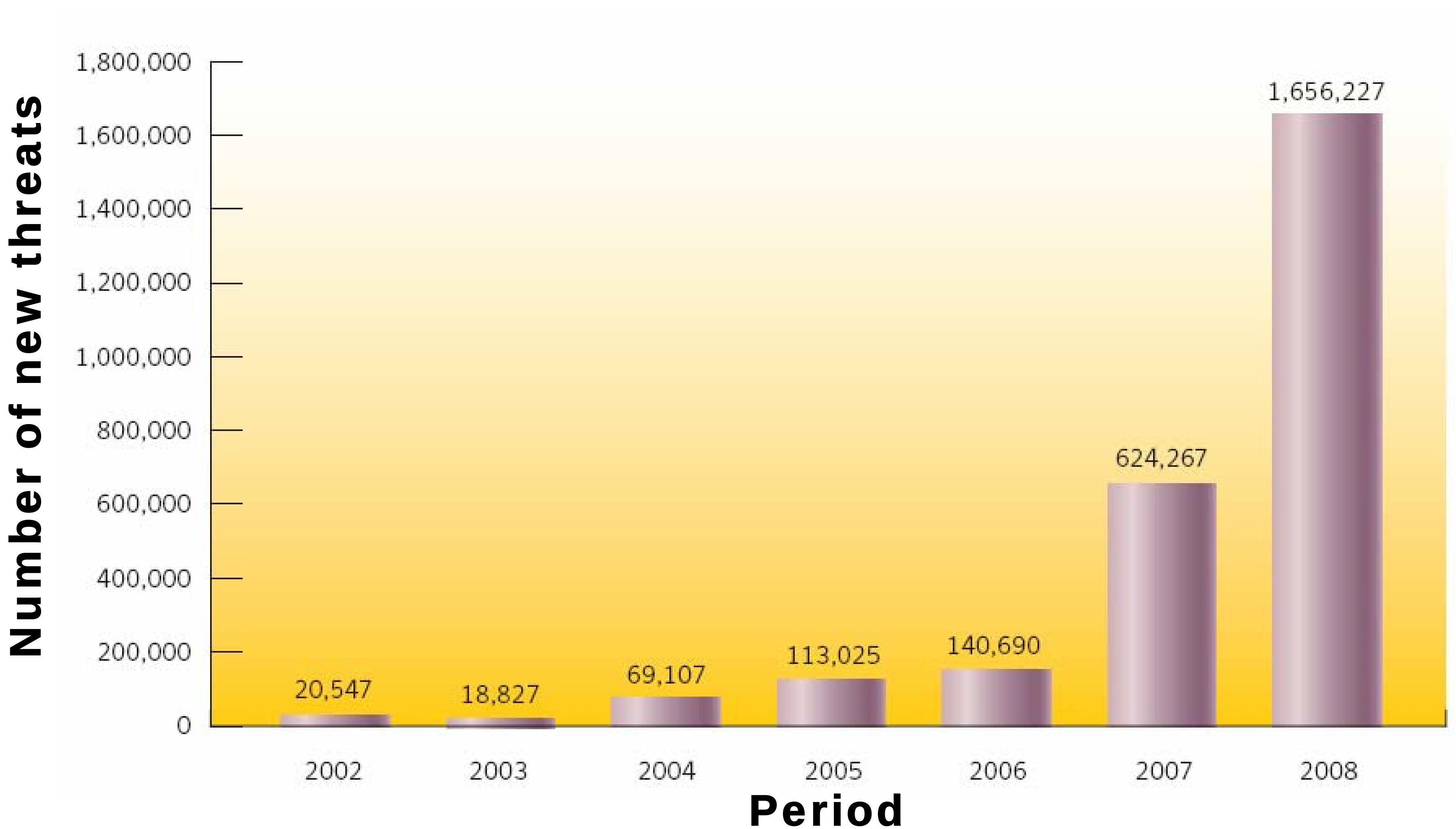
Computer Science Dept.

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]**

Growth of New Malicious Code Threats



(source: Symantec)

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**

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**

BitBlaze Binary Analysis Infrastructure: Challenges

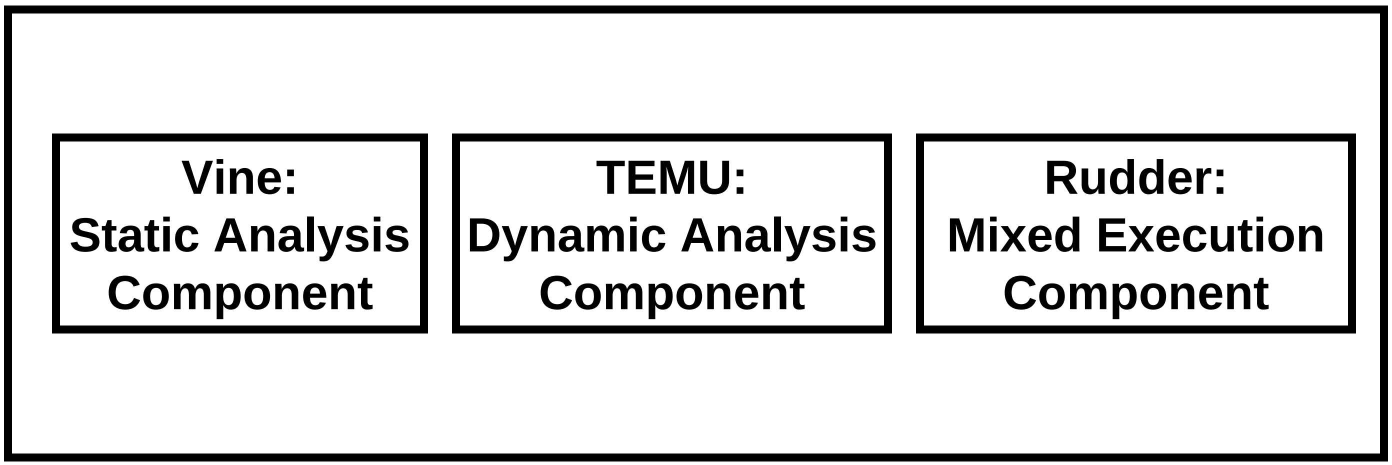
- **Complexity**
  - **IA-32 manuals for x86 instruction set weights over 11 pounds**
- **Lack higher-level semantics**
  - **Even disassembling is non-trivial**
- **Require whole-system view**
  - **Operations within kernel and interactions btw processes**
- **Malicious code may obfuscate**
  - **Code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

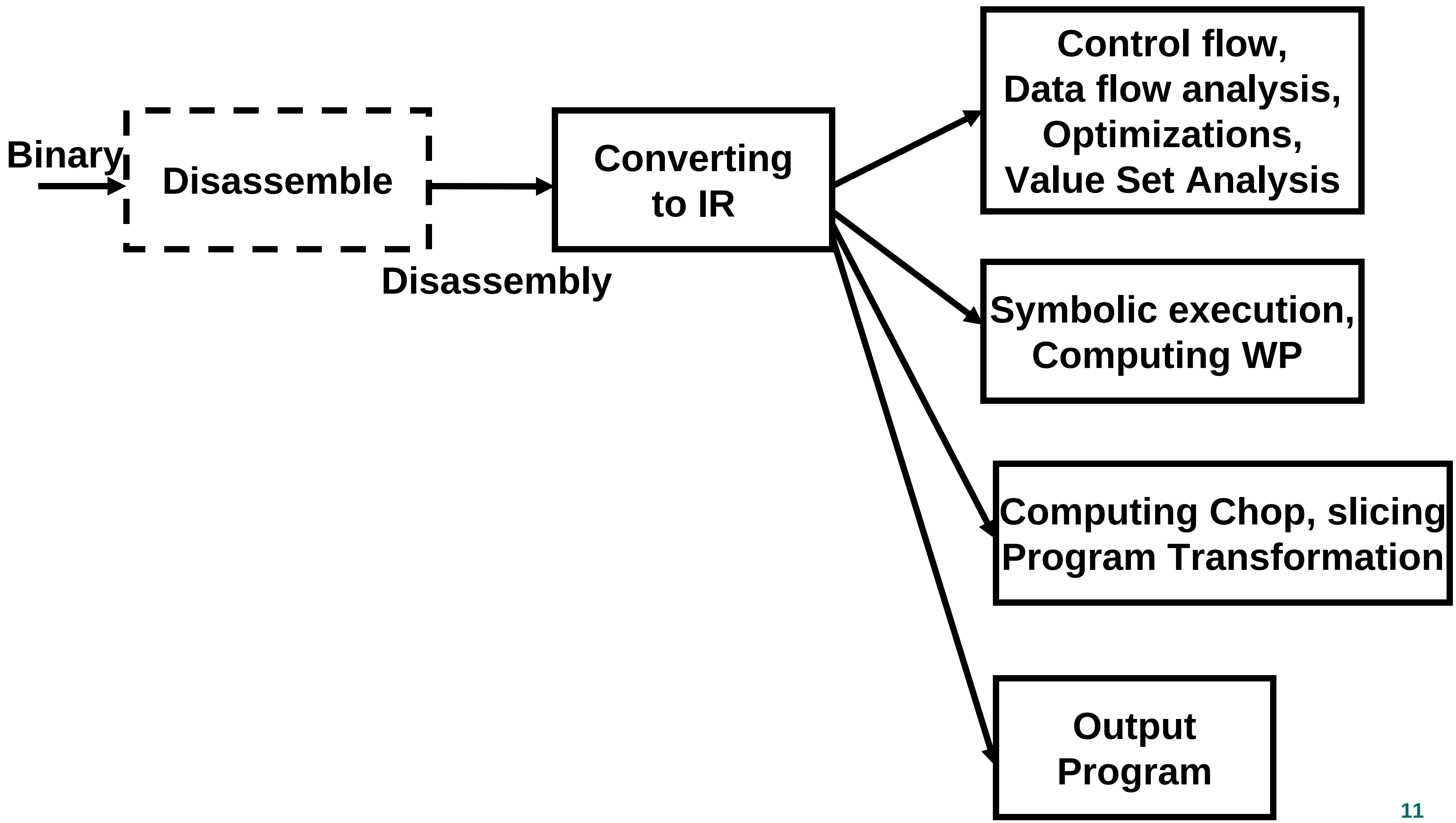
- **The first infrastructure:**
  - **Novel fusion of static, dynamic analysis techniques, and formal analysis techniques such as symbolic execution & abstract interpretation**
  - **Capable of analyzing whole system (including OS kernel)**
  - **Capable of analyzing packed/encrypted/obfuscated code**



BitBlaze Binary Analysis Platform

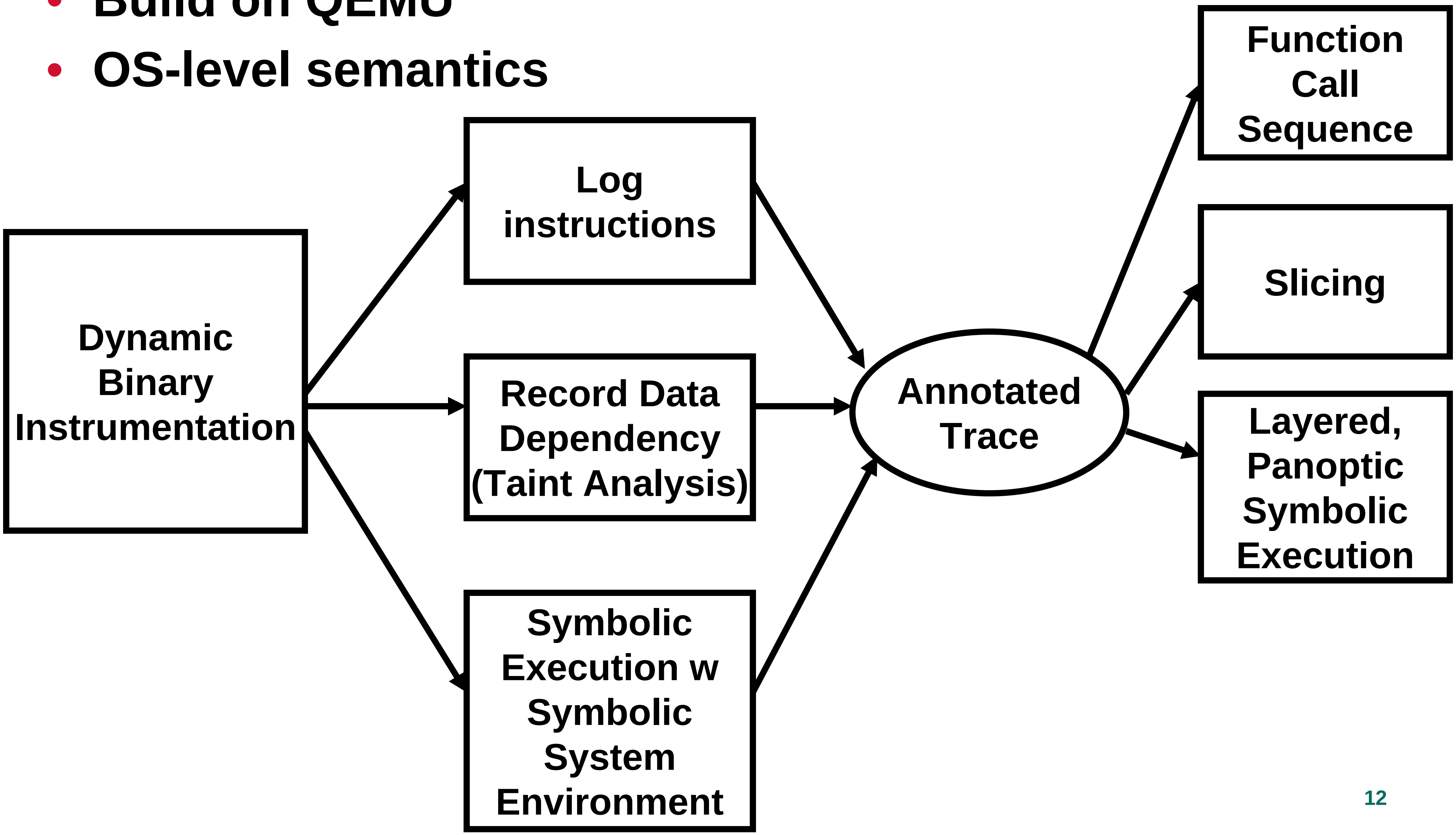
Vine

- **Static analysis component**



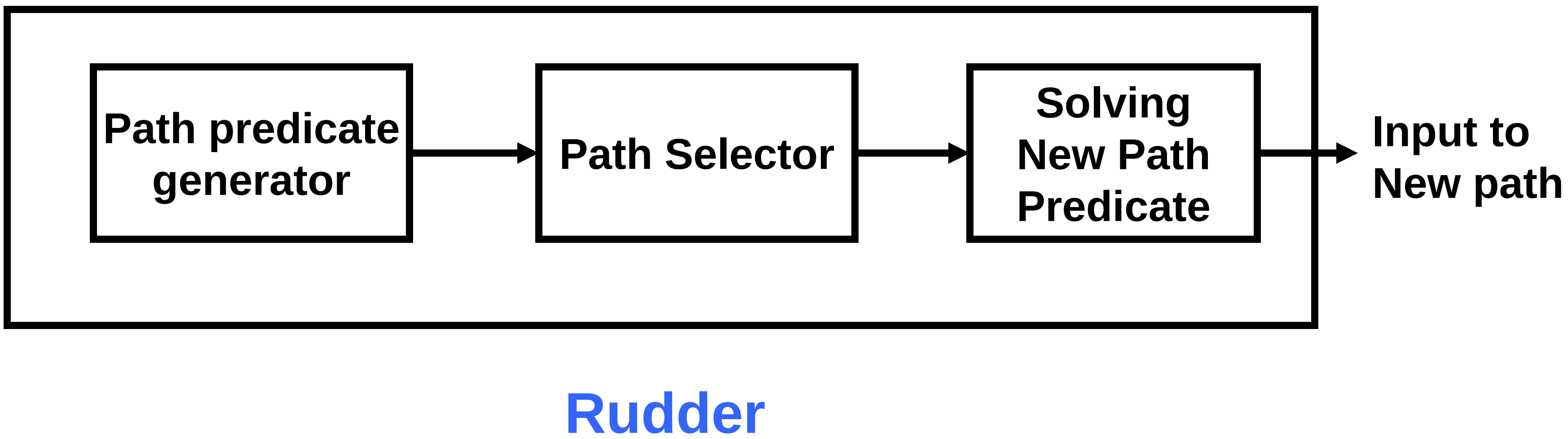
TEMU

- **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**

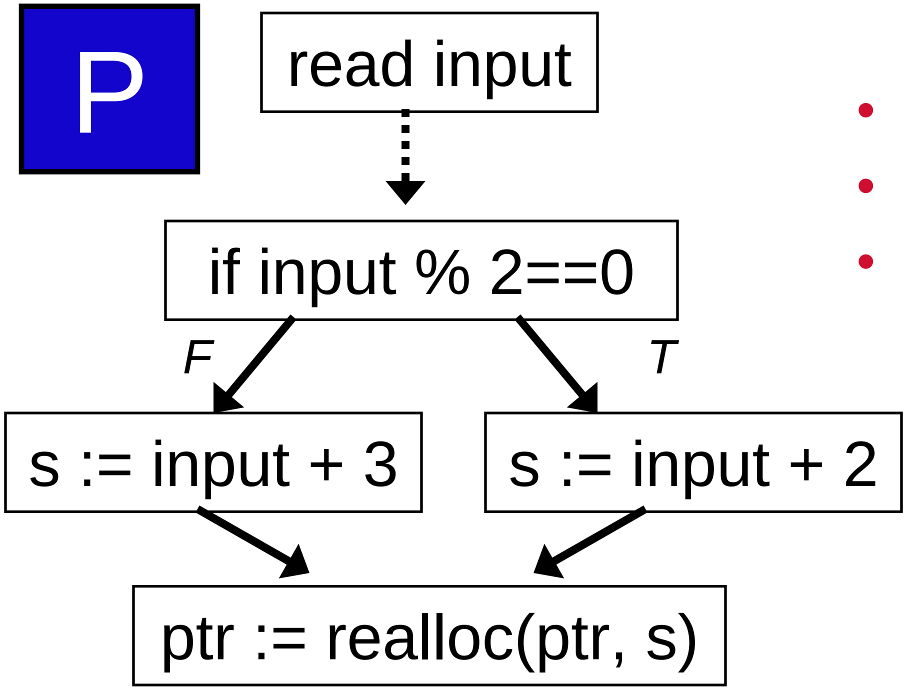
Patch Tuesday

- **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

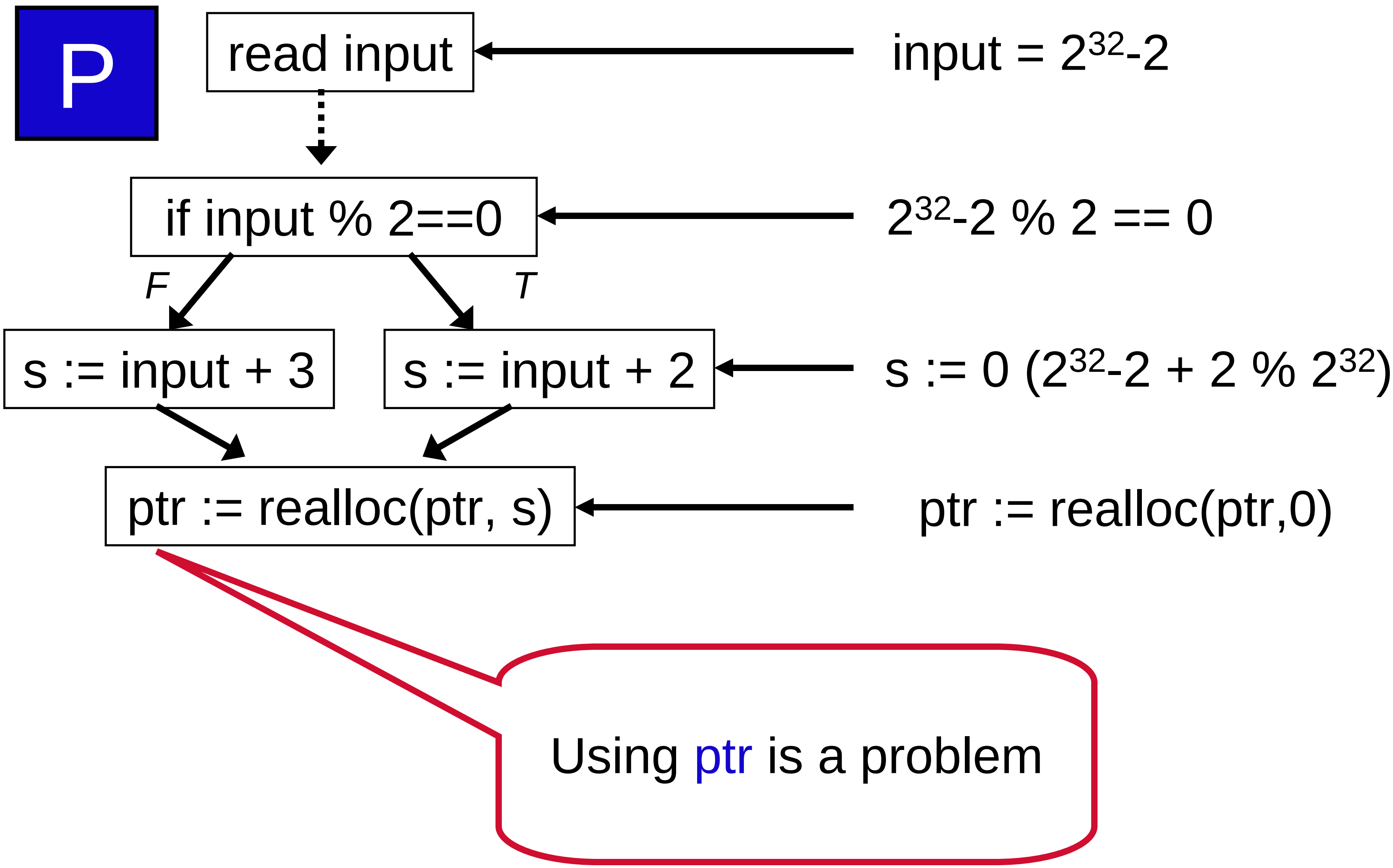
- **Given vulnerable program P, patched program P', automatically generate exploits for P**
- **Why care?**
  - **Exploits worth money**
    - » **Typically \$10,000 - \$100,000**
    - » **Great source of research funding :-)**
  - **Know thy enemy**
    - » **Security of patch distribution schemes?**
    - » **Can a patch make you more vulnerable?**
  - **Patch testing**

Running Example

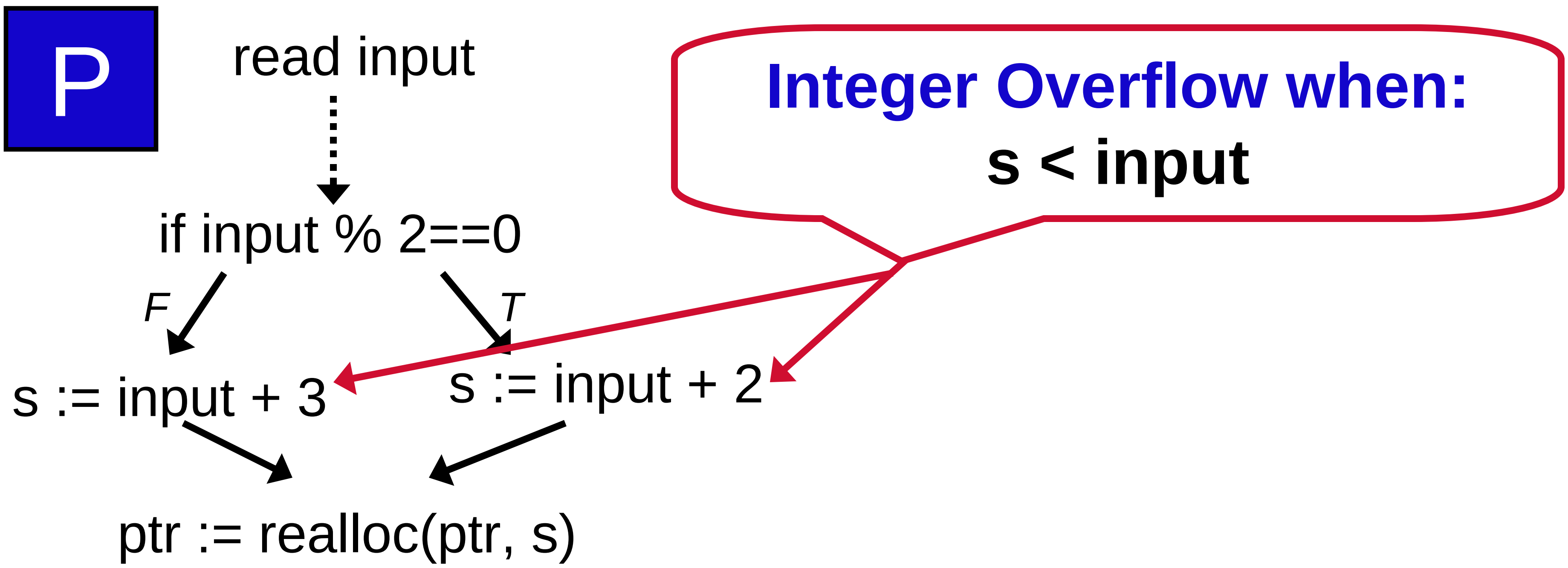


- **All integers unsigned 32-bits**
- **All arithmetic mod 2^{32}**
- **Motivated by real-world vulnerability**

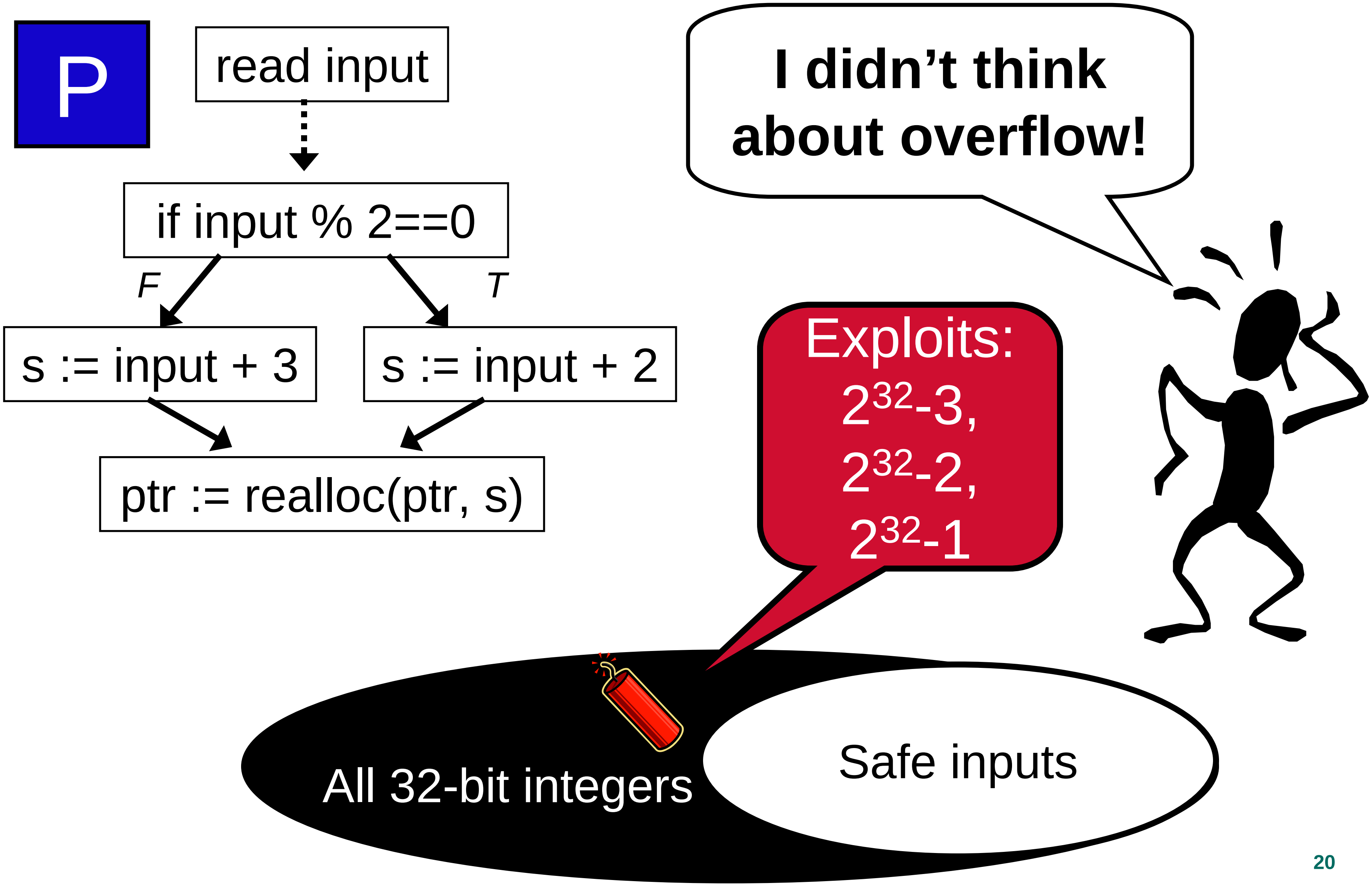
Running Example

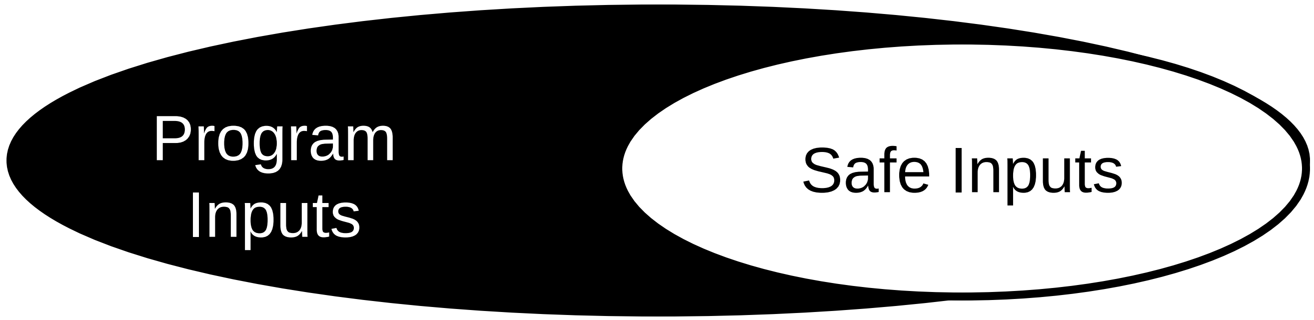


Running Example



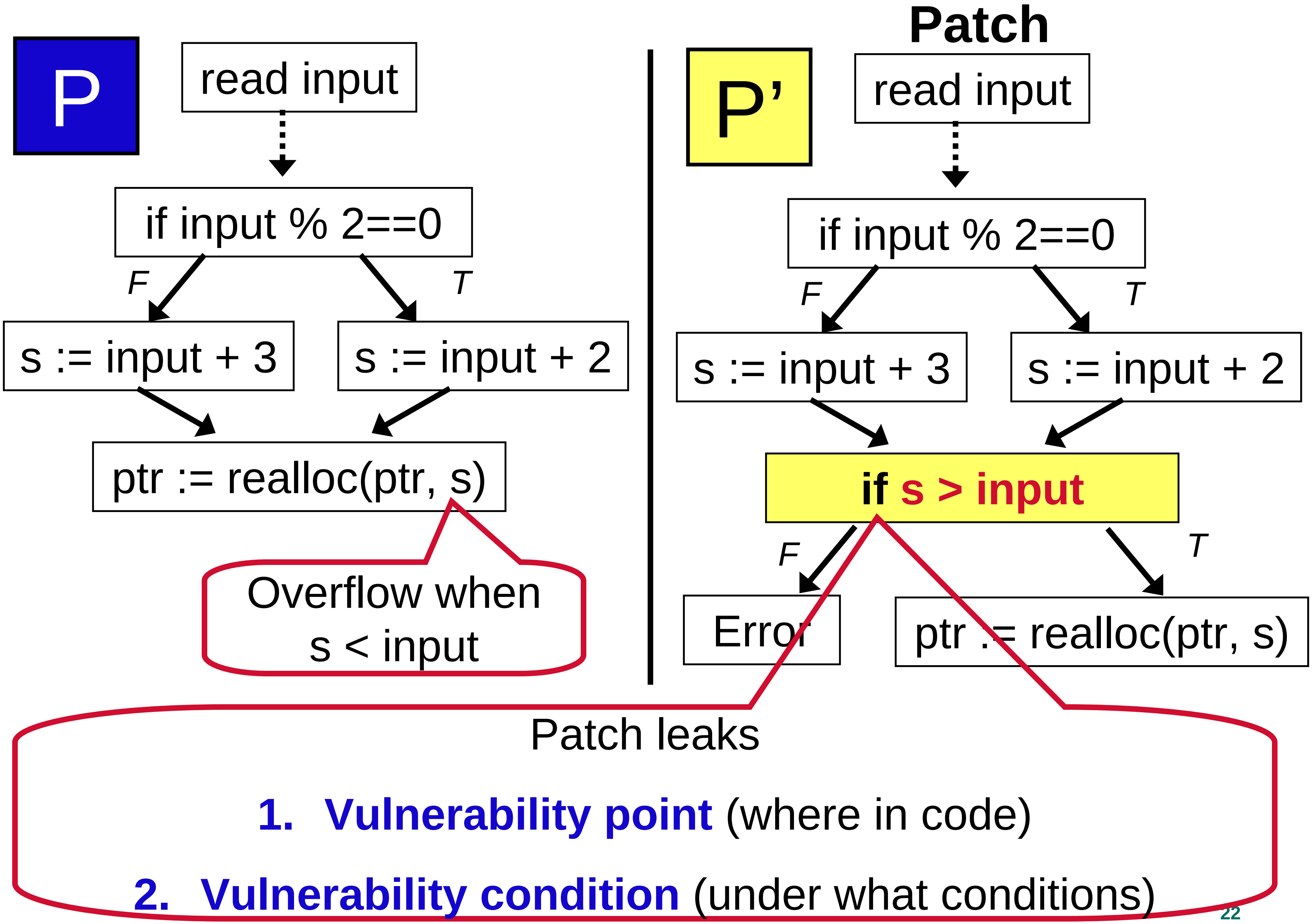
Running Example

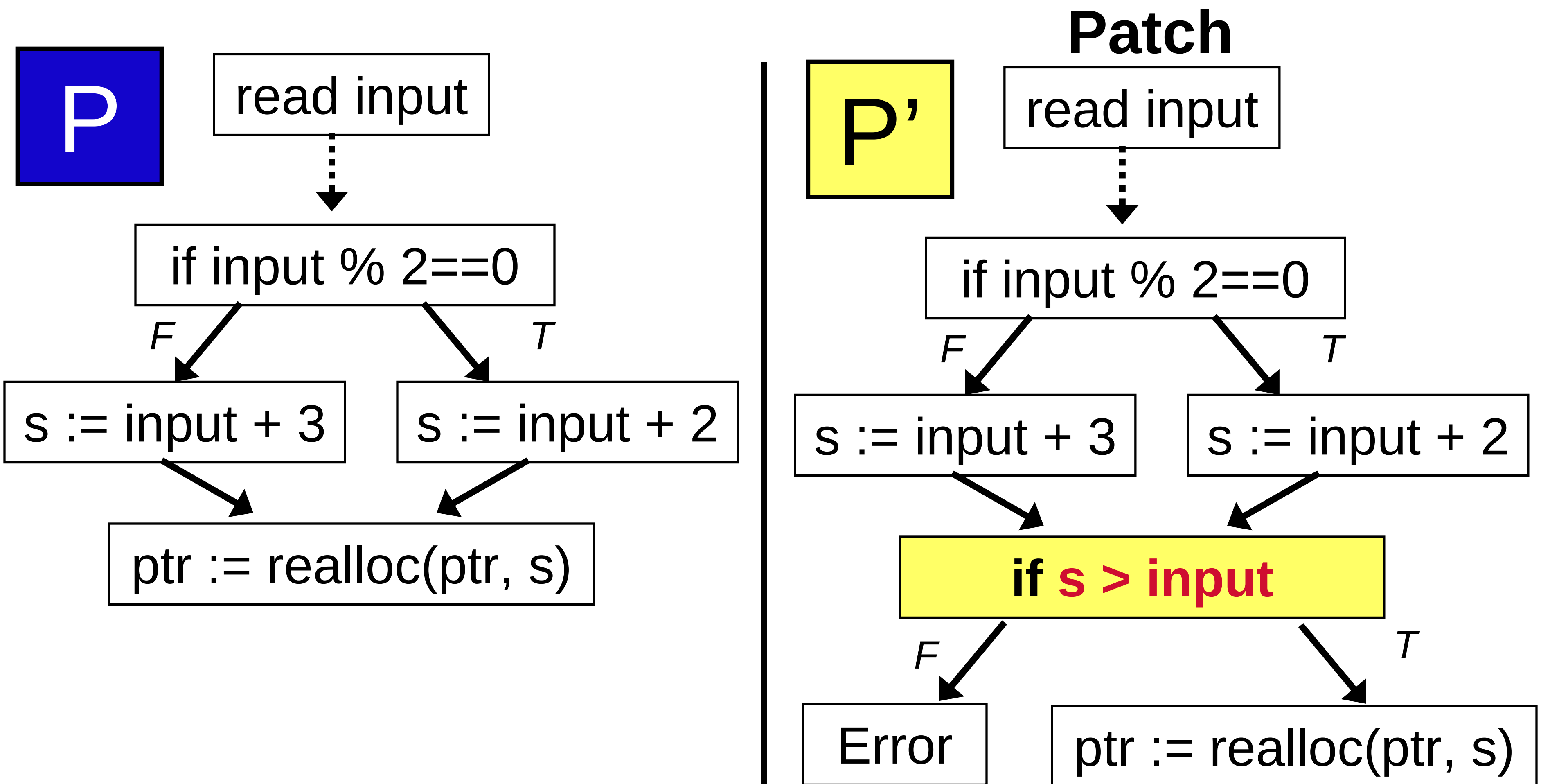




Input Validation Vulnerability

- **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**


P
read input
if input % 2==0
F
T
s := input + 3
s := input + 2
ptr := realloc(ptr, s)
Overflow when
s < input
Patch
P'
read input
if input % 2==0
F
T
s := input + 3
s := input + 2
if s > input
F
T
Error
ptr := realloc(ptr, s)
Patch leaks
1. Vulnerability point (where in code)
2. Vulnerability condition (under what conditions)

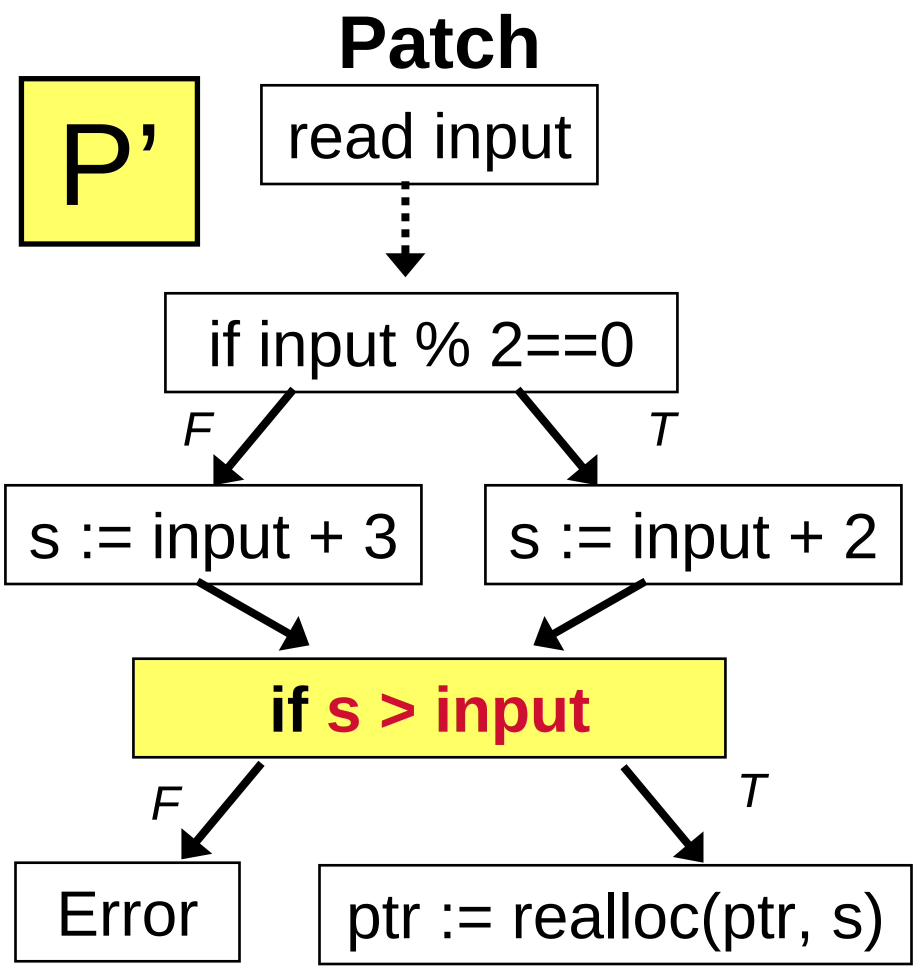


Exploits for P are inputs that fail
vulnerability condition at vulnerability point
(s > input) = false

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

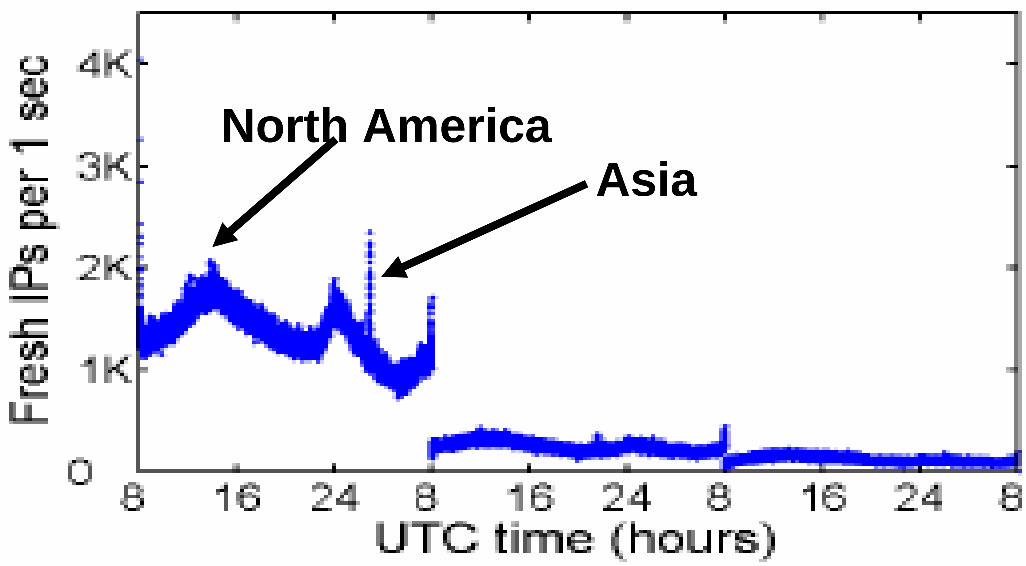
We generate exploits in seconds to minutes

+

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

=

Patch release can create serious threats



Unique IP's contacting Windows Automatic Update

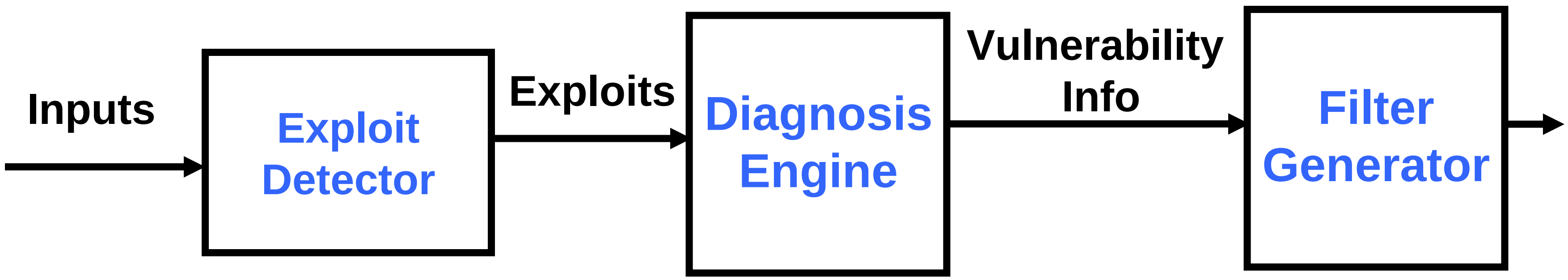
[GKPV06]

Outline

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

Other Security Applications

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



- **Automatic Vulnerability discovery**
  - **Loop extended symbolic execution**
  - **String-enhanced white-box exploration for model extraction**
- **In-depth malware analysis**
- **Others:**
  - **Reverse engineering**
  - **Deviation detection [Best Paper Award]**
  - **Semantic binary diff**

Automatic Vulnerability Discovery (I)

- **BitFuzz**
  - **Smart fuzzing to explore program execution space to find bugs**
  - **Found bugs in real-world programs, e.g., CVE for MS program gdi32.dll**

- **Challenges**
  - **Scalability to large programs**
  - **Inputs with structures**
  - **Programs with loops**
  - **Solving complex constraints**

Automatic Vulnerability Discovery (II)

Advanced symbolic execution for more effective exploration of program execution space:

- **Grammar-aware symbolic execution**
  - **Handle inputs with rich structures**
- **Loop-extended symbolic execution**
  - **Handle programs with loops**
- **New decision procedure for solving complex constraints**
  - **Theory of strings**

Results (I): Vulnerability Discovery

- On 14 Benchmark Applications (MIT Lincoln Labs)
  - Created from historic buffer overflows (BIND, sendmail, wuftp)

- Found at least 1 vulnerability in each benchmark
  - 1 NEW exploit location in sendmail 7 benchmark

- Highly effective for testing:
  - Over 60% candidates were real attacks.
  - **20** real vulnerabilities out of **32** candidates exploits.

Results (II): Real-world Vulnerabilities

- Diagnosis and Discovery 3 Real-world Case Studies
  - SQL Server Resolution [Slammer Worm 2003]
  - GDI Windows Library [MS07-046]
  - Gaztek HTTP web Server
- Diagnosis Results
  - Results precise and field level
- Discovery Results: Found **4** buffer overflows in **6** candidates
  - 1 new exploit location for Gaztek HTTP server

Program	Buffer size (bytes)	Condition for overflow
GHttpd (1)	220	`URI.len > 172`
NEW GHttpd (2)	208	`URI.len > 133`
SQL Server	128	`DBName.len > 64`
GDI	4096	`(2·INP[19:18])»2 < 0`

Results (III): Code Coverage

- Qualitative Measurement
- New loop based symbolic constraints: 270 in 17 targets
  - On an average 15 new constraints become symbolic

Program	Input Format	Initial Input	Exploit Input	Bug / Candidate	Time (s)	Loop-Dep. Conditions
BIND 1	DNS QUERY	104 bytes, RDLen=48	RDLen=16	1/5	2511	16
BIND 2	DNS QUERY	114 bytes, RDLen=46	RDLen=30	1/4	2155	12
BIND 3	DNS IQUERY	39 bytes, RDLen=4	RDLen=516	1/2	586	13
BIND 4	DOMAINNAME	"web.foo.mit.edu"	"web.foo.mit.edu" (64 times)	1/1	4464	52
Sendmail 1	Byte Array	"<><><>"	"<>" (89 times)	4/5	672	1
Sendmail 2	struct passwd (Linux)	("","root",0,0,"root","","")	("","root",0,0,"rootroo","","")	1/1	526	38
Sendmail 3	$[String]^N$	$[\text{"a=\textbackslash n"}]^2$	$[\text{"a=\textbackslash n"}]^{59}$	1/4	626	18
Sendmail 4	Byte Array	"aaa"	"a" (69 times)	1/1	633	2
Sendmail 5	Byte Array	"\\\"	"\" (148 times)	3/3	18080	6
Sendmail 6	OPTION∘' '∘ARG	"-d aaaaaaaaaa-2"	"-d 4222222222-2"	1/1	676	11
Sendmail 7	DNS Response Fmt	TXT Record : "aaa"	Record : "a" (32 times)	1/1	237	16
WuFTP 1	String	"aaa"	"a" (9 times)	2/2	483	5
WuFTP 2	PATH	"aaa"	"a" (10 times)	1/1	197	29
WuFTP 3	PATH	"aaa"	"a" (47 times)	1/1	109	7
GHttpd	Method∘URI∘Version	"GET /index.html HTTP/1.1"	"GET "+188 bytes + " HTTP/1.1"	2/2	1562	41
SQL Server	Command∘DBName	x04 x61 x61 x61	x04 x61(194 bytes)	1/3	205	1
GDI	(Not required)	1014 bytes, INP[19:18]=0x0182	INP[19:18]=0x4003	1/1	353	2

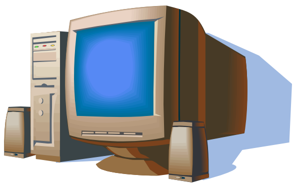
NEW

Automatic Model Extraction

- **Automatic model extraction**
  - **E.g., identifying vulnerability in web browsers' security policy**

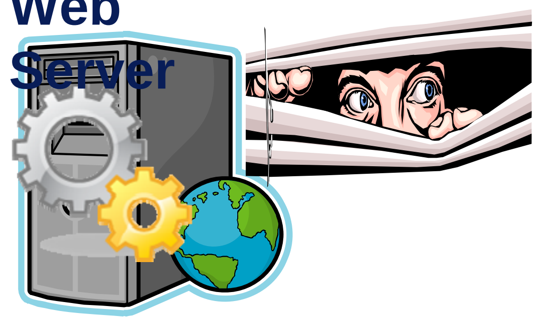
- **Automatic grammar/protocol extraction**
  - **Automatic grammar-aware symbolic execution and grammar extraction combine seamlessly and enhance each other**

Symbolic Execution: Path Predicate



GET / HTTP/1.1

Web Server



x86 instructions

```
MOV (%esi), %al
MOV $0x47, %bl
CMP %al, %bl
JNZ FAIL
MOV 1(%esi), %al
MOV $0x45, %bl
CMP %al, %bl
JNZ FAIL
…
```

Intermediate Representation (IR)

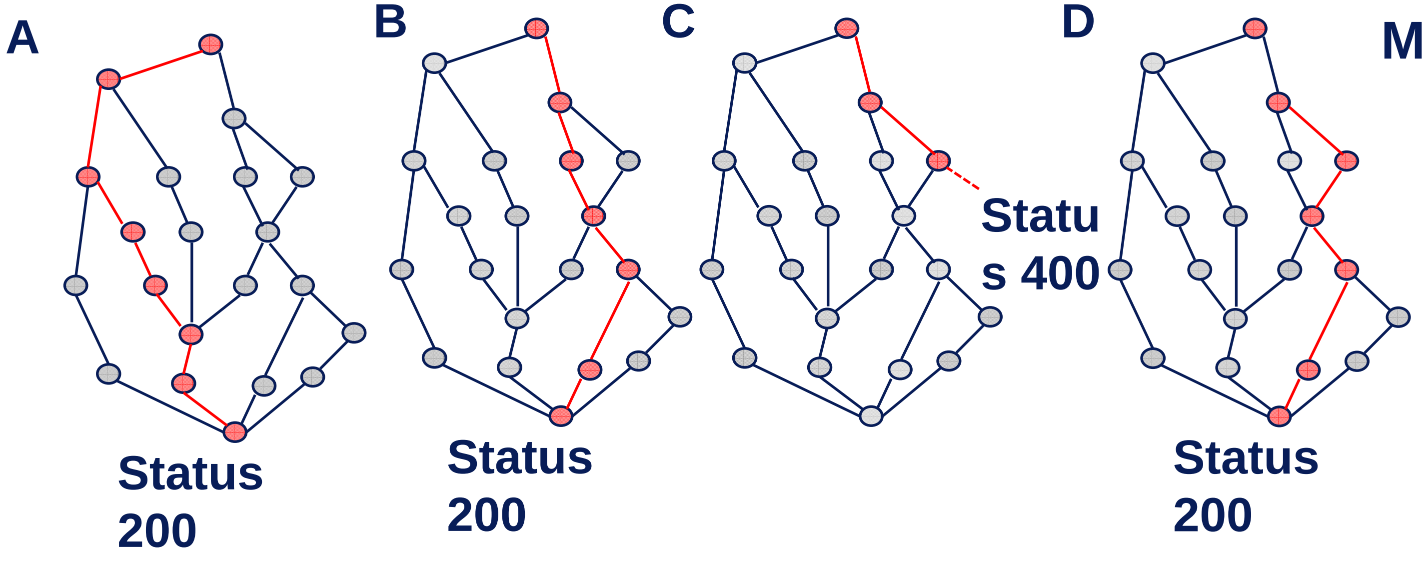
```
AL = INPUT[0]
BL = ‘G’
ZF = (AL == BL)
IF(ZF==0)JMP(FAIL)
AL = INPUT[1]
BL = ‘E’
ZF = (AL == BL)
IF(ZF==0)JMP(FAIL)
…
```

Path predicate

(INPUT[0] == ‘G’)
^
(INPUT[1] == ‘E’)
^
…

White-Box Model Extraction

- **White-box exploration**
  - **Obtain path predicate using symbolic input**
  - **Reverse condition in path predicate**
  - **Generate input that traverses new path**
  - **Iterate until user-specified timeout expires**
- **Model: disjunction of path predicates**



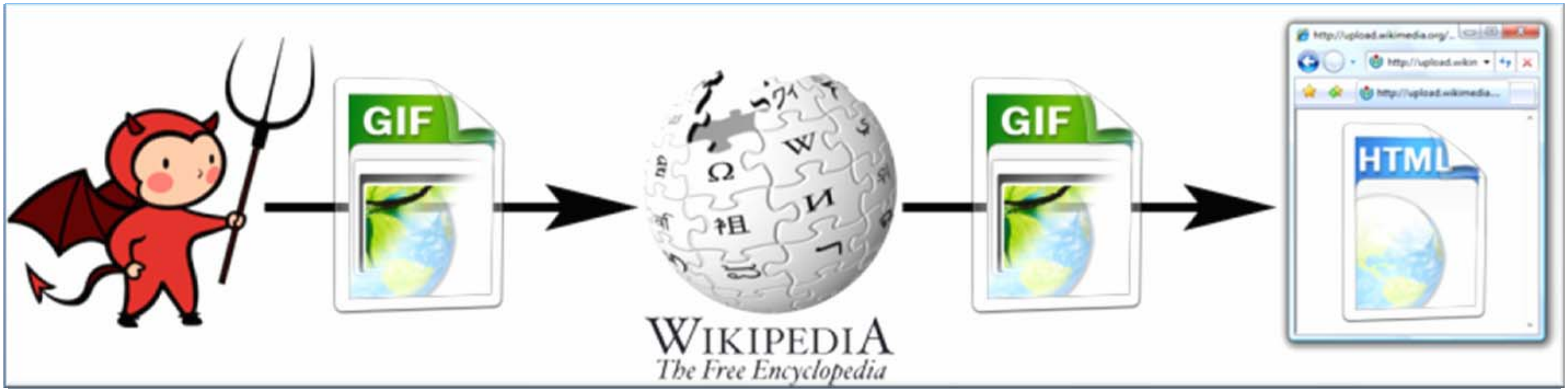
$M_{200} = A \vee B \vee D$

Extracting Content Sniffing Algorithms in Browsers

Browser	Signature for image/gif
Internet Explorer 7	**(strncasecmp(DATA,“GIF87”,5) == 0) || (strncasecmp(DATA,“GIF89”,5) == 0)**
Firefox 3	**strncmp(DATA,“GIF8”,4) == 0**
Safari 3.1	**N/A**
Google Chrome	**(strncmp(DATA,“GIF87a”,6) == 0) || (strncmp(DATA,“GIF89a”,6) == 0)**

Browser	Signature for image/jpeg
Internet Explorer 7	**DATA[0:1] == 0xffd8**
Firefox 3	**DATA[0:2] == 0xffd8ff**
Safari 3.1	**DATA[0:3] == 0xffd8ffe0**
Google Chrome	**DATA[0:2] == 0xffd8ff**

Content Sniffing XSS Attacks

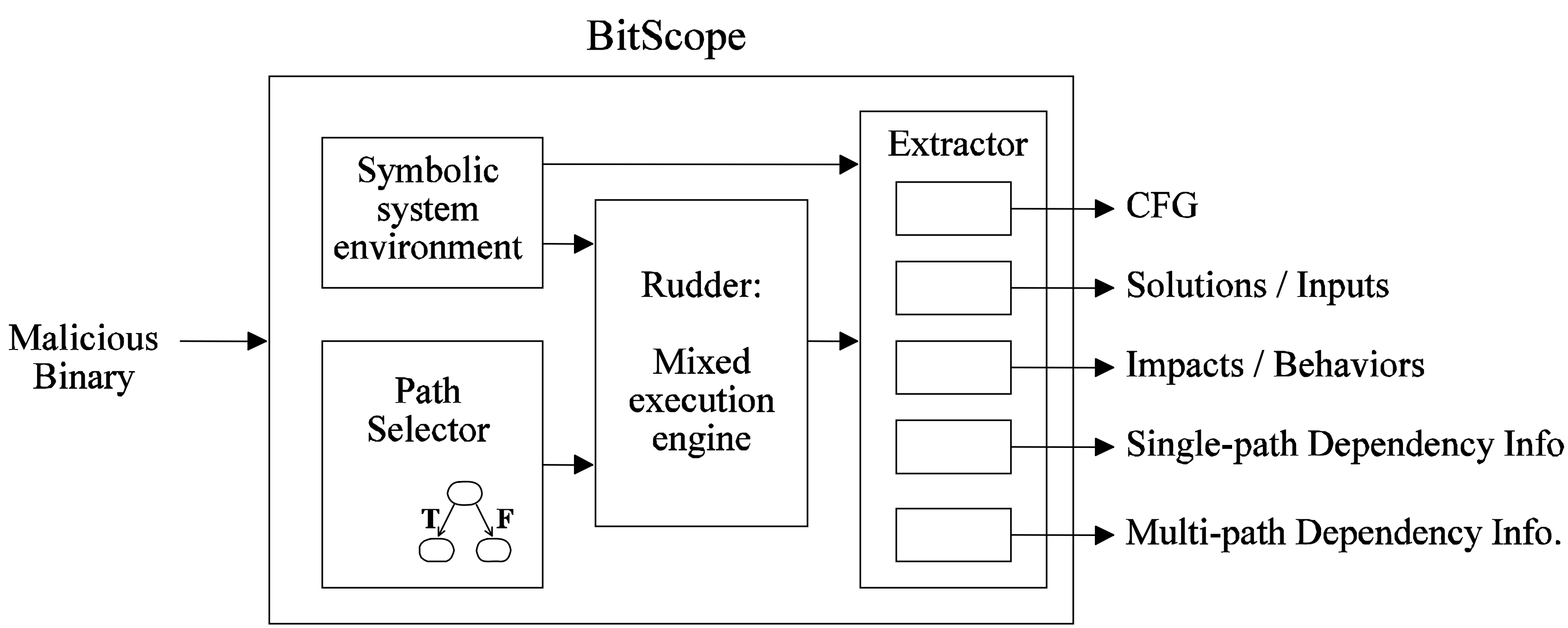


In-depth Malware Analysis

- 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**
  - **Extracting behaviors**

- **Parallel architecture for high-volume malware analysis**

- **Public service:**
  - **Submit malware samples and get results back**

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, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, Prateek Saxena, Heng Yin