

Analysis and Defense of Vulnerabilities in Binary Code

David Brumley
Carnegie Mellon University
dbrumley@cmu.edu

1

B
Buggy Program



P
Patched New Program

Buggy programs are one of the leading causes of hacked computers

“Regularly Install Patches”
– Computer Security Wisdom

2

Patches Can Help Attackers

– *Evil David*



Evil David

3

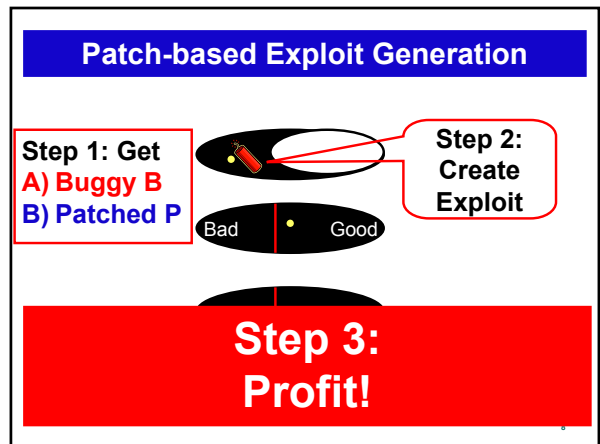
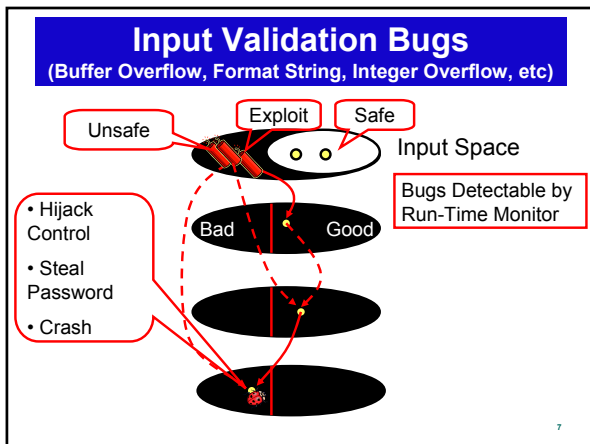
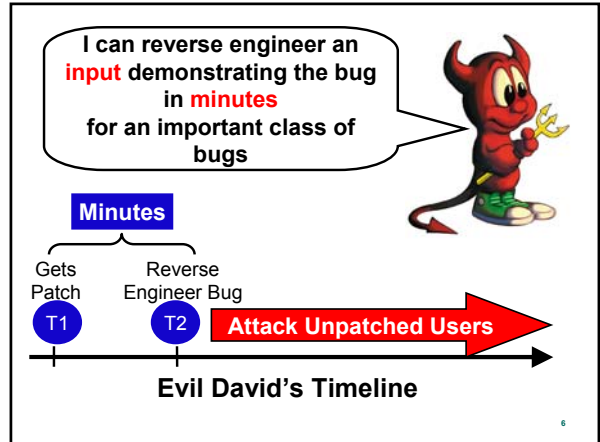
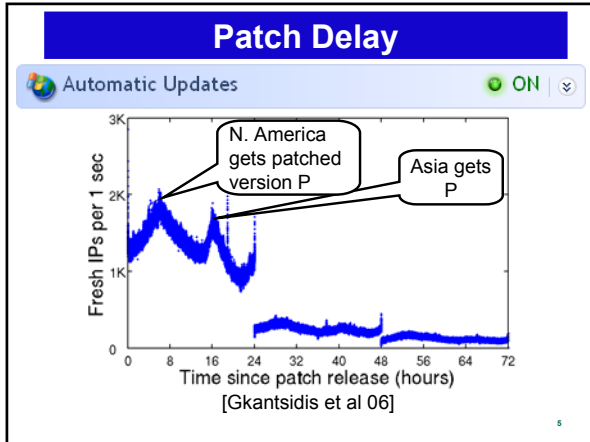


Evil David

Delayed Patch Attack



4



Exploits Can Help Defenses

– *Good David*

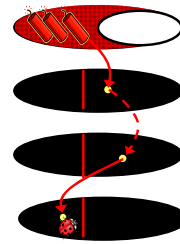


Good David

9

Exploits *Demonstrate* Bugs

Goal:
Recognizer
for unsafe
inputs

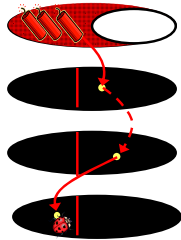


Given:
1. Exploit
2. B

10



Goal:
Recognizer
for unsafe
inputs



Given:
1. Exploit
2. B

11

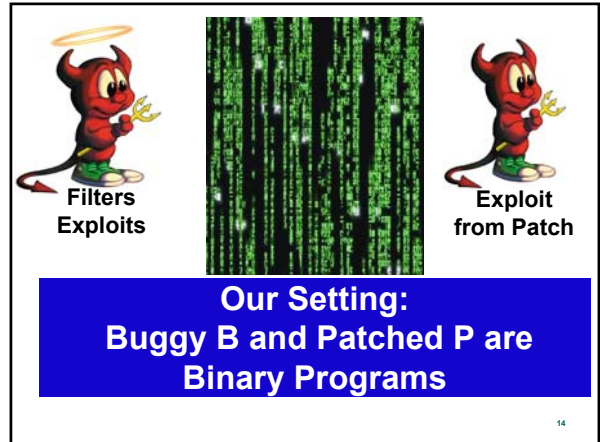
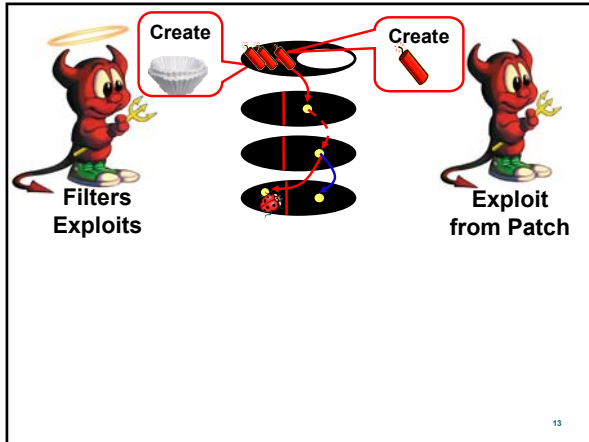


Good David


Creates Filters to
Defend Against
Exploits

Filters are a core component of
widely used defenses from
Symantec, Trend Micro, etc.

12



**Vine:
Security-Relevant
Binary Program Analysis Architecture**



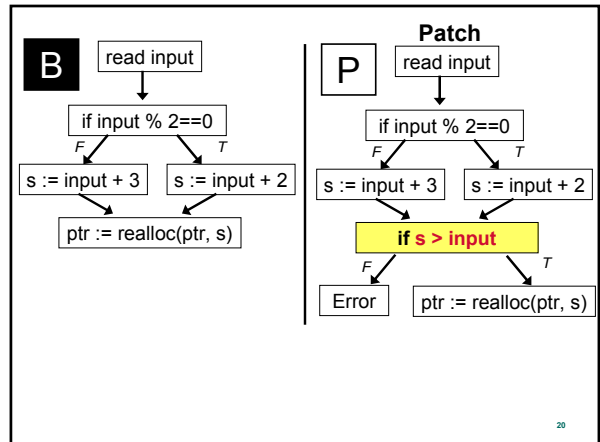
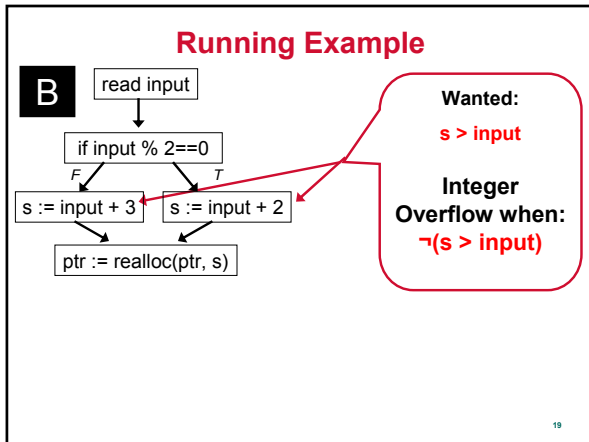
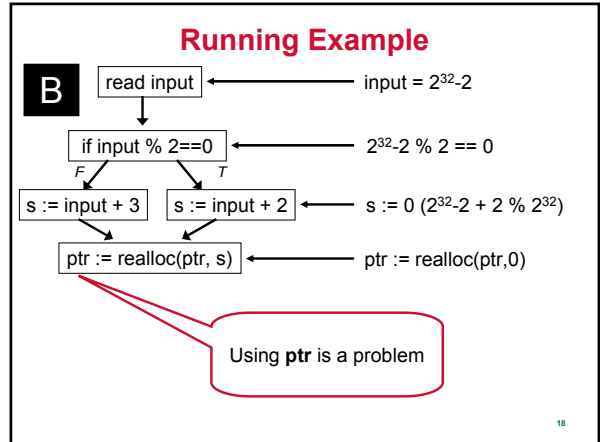
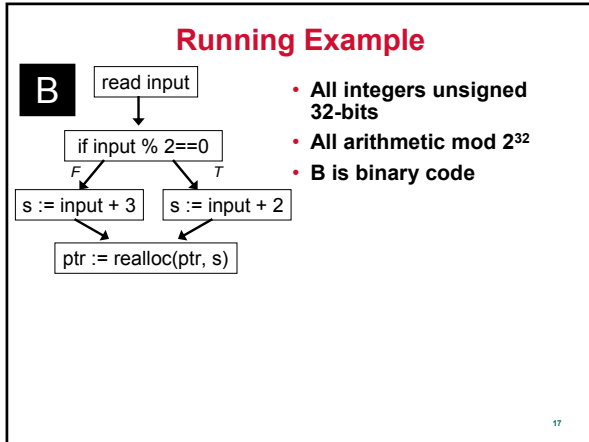
- Binary code is everywhere
- Security of the code you run (not just the code compiled)

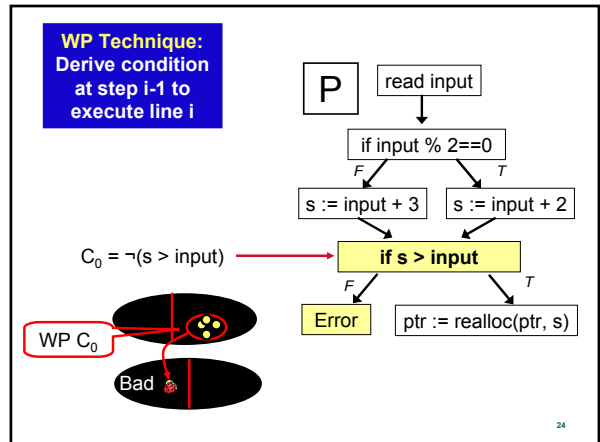
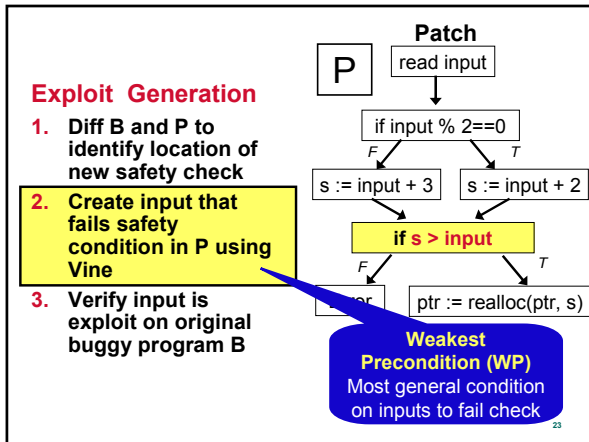
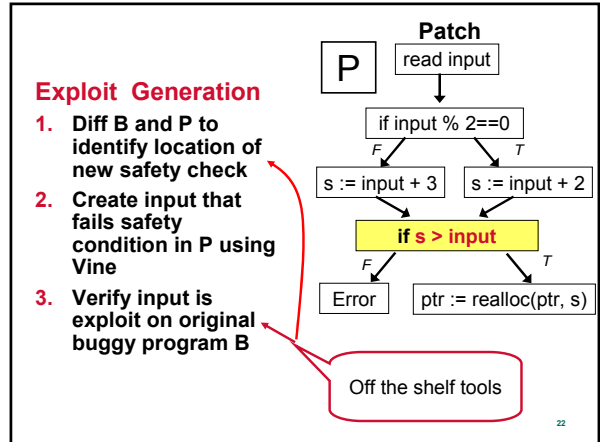
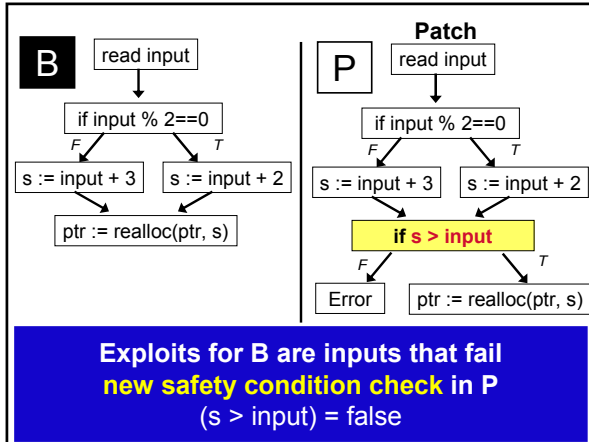
15

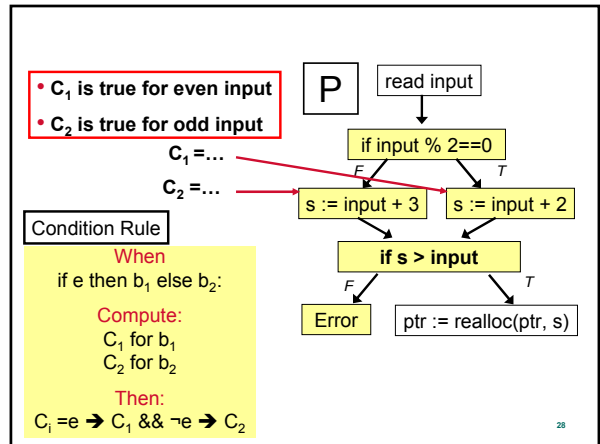
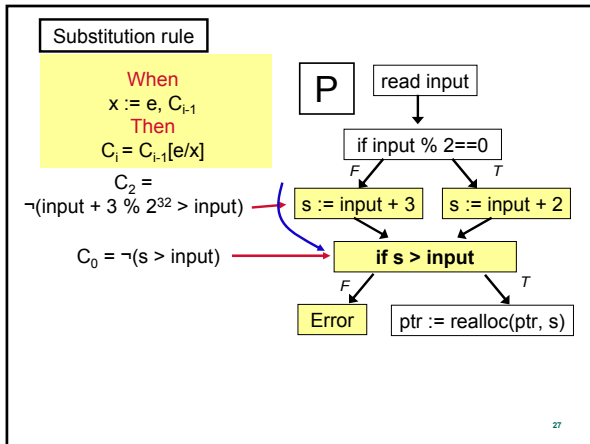
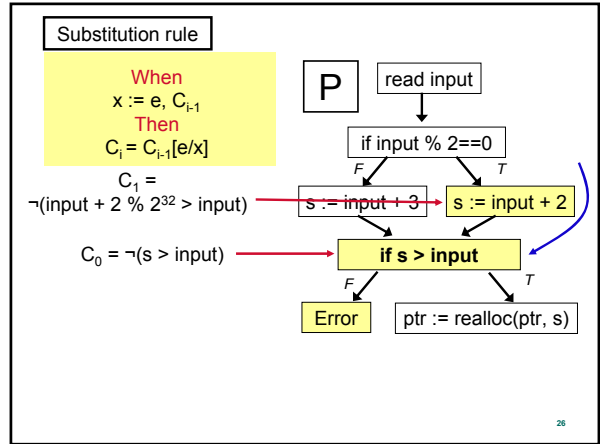
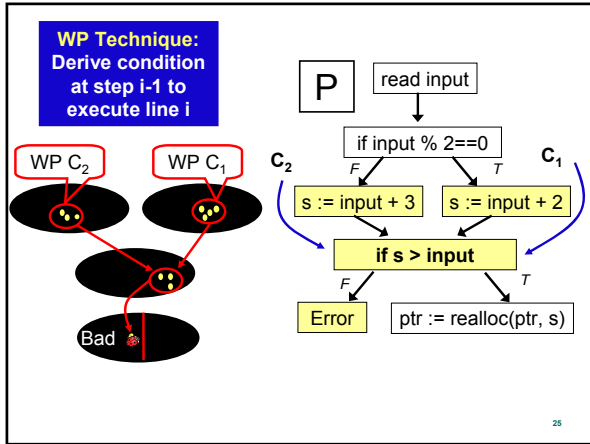
Talk Outline

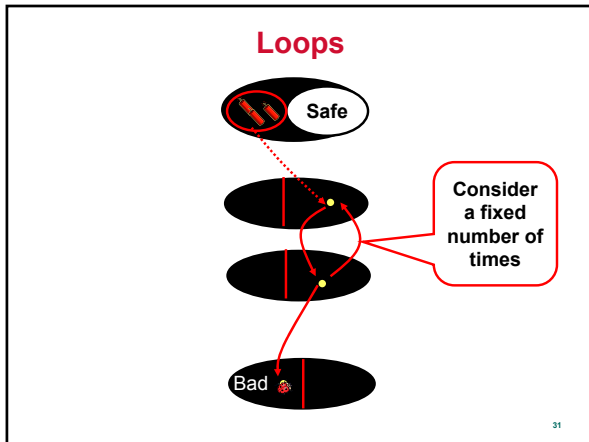
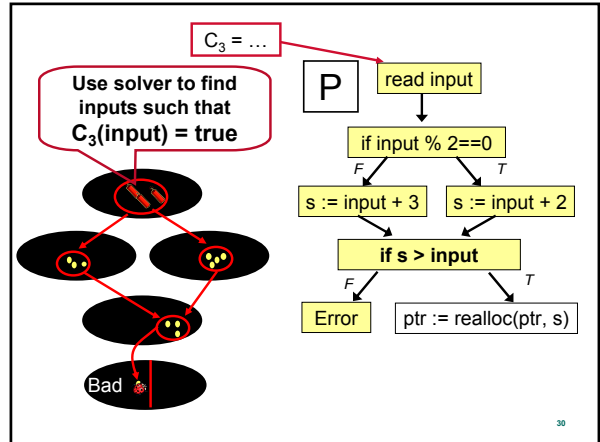
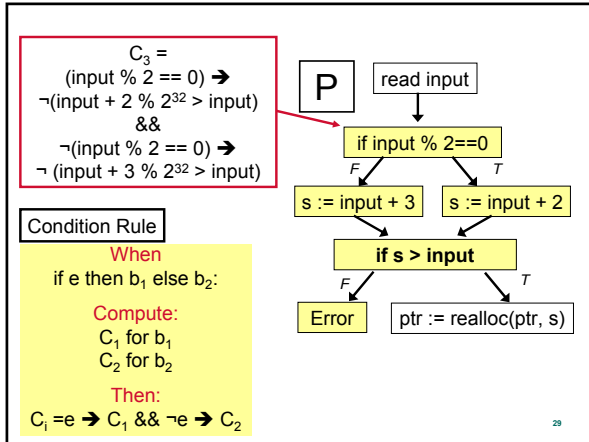
1. **Automatic Patch-Based Exploit Generation**
 - Generate inputs that execute specific line of code (weakest precondition)
 - Results
2. **Automatic Input Filter Generation**
 - New program analysis approach to filter generation
 - Filters have accuracy guarantees
3. **Vine Platform for Binary Analysis** Vine
 - Infrastructure
4. **Questions and Answers**

16









Two Problems with Classic Weakest Precondition:

1. Not suitable for programs with gotos (unstructured programs)
2. Final C is **exponential** in size
 - Due to substitution for assignment Substitution rule
 - Duplication of condition in branches Condition Rule

$$C_3 =$$

$$(input \% 2 == 0) \rightarrow$$

$$\neg(input + 2 \% 2^{32} > input)$$

$$\&\&$$

$$\neg(input \% 2 == 0) \rightarrow$$

$$\neg(input + 3 \% 2^{32} > input)$$

In Brumley et al 07:

1. Generalized WP for programs with goto's (unstructured programs)
2. Prove $O(n^2)$ size for condition C (n = number of statements)
 - Proof generalizes [Leino05]

33

Putting it All Together

1. Diff B and P to identify location of new safety check
2. Create input that fails safety condition
 - A. Lift to Vine
 - B. Calculate weakest precondition for vulnerability
 - C. Use solver to find inputs that satisfy condition : $C(\text{input}) = \text{true}$
3. Verify **input** is exploit on original buggy program B

34

Exploit Generation Results

| ASPNet_Filter | Information Disclosure | 29 sec |
|-----------------|------------------------|---------|
| GDI | Hijack Control | 119 sec |
| PNG | Hijack Control | 131 sec |
| IE COMCTL32 (B) | Hijack Control | 378 sec |
| IGMP | Denial of Service | 186 sec |

- No public exploit for 3 out of 5
- Exploit unique for other 2


35

Demo of ASPNet_Filter Info Disclosure

36

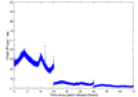
When could technique fail?

- Cannot solve condition C
- Not enough loop iterations
- Timing vulnerabilities
- etc.



However, security must conservatively estimate attackers capabilities
(We don't know for future bugs)

Current Delayed Patch Distribution Insecure



38




New Research Problem: Prevent Patches From Helping Attackers

Ideas?

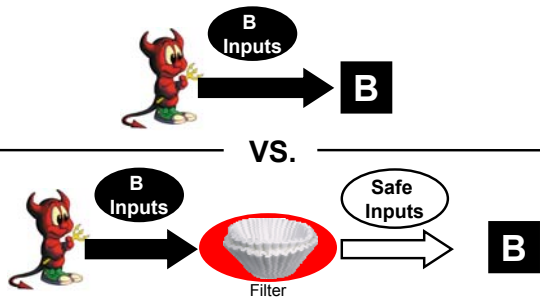
- **Code Analysis: Obfuscate patches**
 - Prevents diffing in our approach, no changes to current update schemes
 - Con: May slow down program, may be insufficient
- **Crypto: Encrypt patch initially, broadcast decryption key**
 - Fair: Everyone applies patch simultaneously
 - Con: Which patches to encrypt? Requires changes to current update schemes, offline hosts?
- **Others**

39

Talk Outline

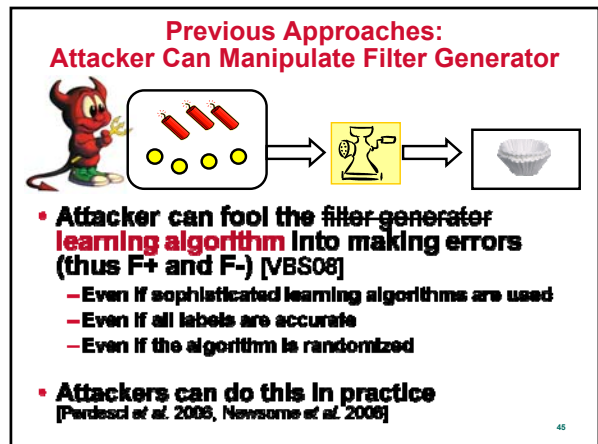
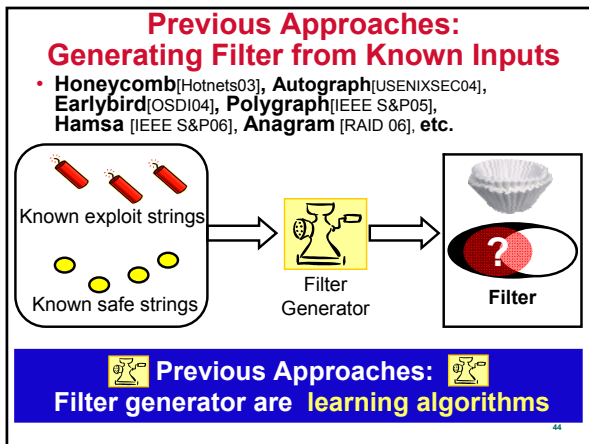
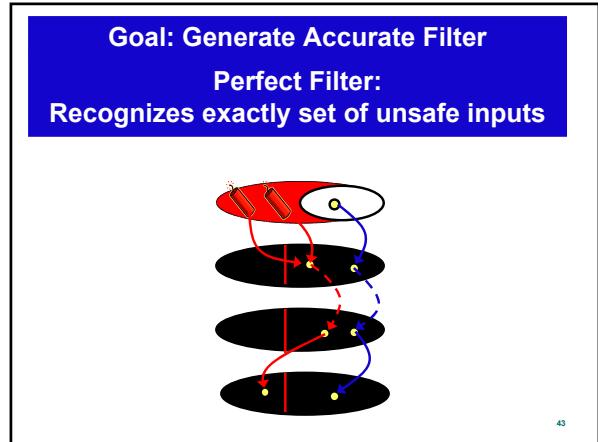
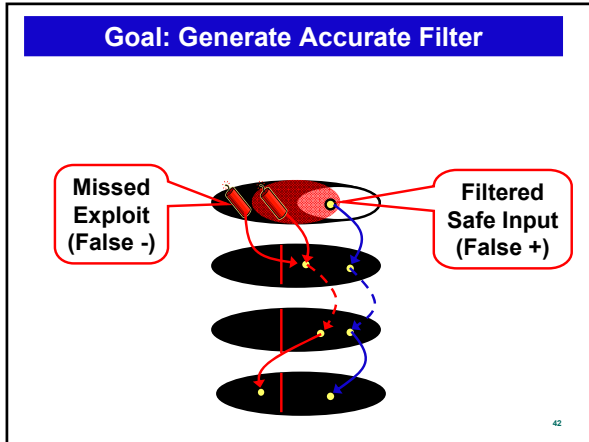
- 1. Automatic Patch-Based Exploit Generation**
 - Generate inputs that execute specific line of code (weakest precondition)
 - Results
- 2. Automatic Input Filter Generation**
 - New program analysis approach to filter generation
 - Filters have accuracy guarantees
- 3. Vine Platform for Binary Analysis** 
 - Infrastructure
- 4. Questions and Answers**

40



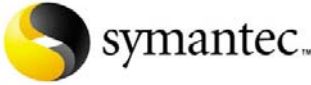

Goal: Generate Accurate Filter

41



Filter?

Filter generation in practice?


46



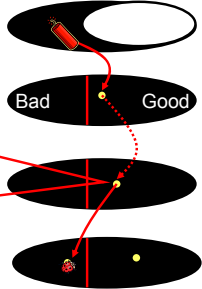
Good David

Can we create filters with accuracy guarantees automatically?
[Brumley 2005-present]

47

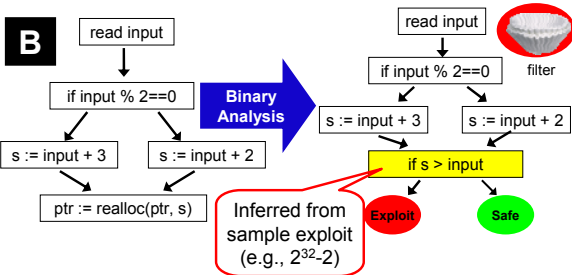
Given { **B**  Single sample exploit

Infer safety condition from execution of sample exploit



48

Perfect Filter: Recognizer of Vulnerability Language
[Brumley et al 06,07]



49

Q: Can we create filters with accuracy **guarantees** automatically?
[Brumley 2005-present]

Yes!

Perfect* by construction

Filter

Evil David

Regardless of the attacker

Initial Filter

```

    graph TD
      A[read input] --> B{if input % 2 == 0}
      B --> C[s := input + 3]
      B --> D[s := input + 2]
      C --> E{if s > input}
      D --> E
      E --> F((Exploit))
      E --> G((Safe))
  
```

Optimized Filter

```

    graph TD
      A[read input] --> B{if input < 2^32-3}
      B --> C((Exploit))
      B --> D((Safe))
  
```

Optimize

Only care about check

Filter Optimization = Code Optimization
(requires binary analysis, not just binary rewriting)

Optimizations help program analysis

2x speedup for exploit generation by optimizing before WP

Exploits are inputs where $\neg(\text{input} < 2^{32}-3) = \text{true}$

```

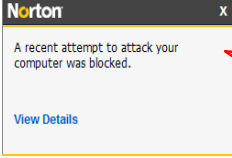
    graph TD
      A[read input] --> B{if input < 2^32-3}
      B -- F --> C[Error]
      B -- T --> D[...]
  
```

Filter Generation Results
[Brumley et al 06, 07]

| Name | Accuracy | Gen Time | Pct Buggy Program | Eval Time (μs) |
|----------|----------|----------|-------------------|----------------|
| MOBB 10 | Perfect | 1.5s | 16% | 13 |
| COMCTL32 | Perfect | 1.2s | 8% | 31 |
| MOBB 19 | Perfect | 9.4s | 4% | 6 |

Web Page (HTML) → Safe → Render in Browser

Filters and techniques used by Symantec/Norton (> 2,000,000 install base)

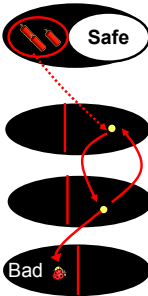


Accuracy guaranteed by my work

54

Other Filters Based On Perfect Filter

Perfect Filter



```

graph TD
    A[read input] --> B{if input < 2^32-3}
    B -- F --> C((Exploit))
    B -- T --> D((Safe))
        
```

May be Turing Complete

56

Other Filters Based On Perfect Filter

Faster

Perfect Filter

May be Turing Complete

↓ Sound Approximation

Finite Execution,
Boolean Predicate using WP
[Brumley07]

Filter over finite domain

↓ Sound Approximation

Exploits are inputs where
 $\neg(\text{input} < 2^{32}-3) = \text{true}$

Filter

Sound = No false +

57

Other Filters Based On Perfect Filter

Faster

Perfect Filter

May be Turing Complete

↓ Sound Approximation

Finite Execution,
Boolean Predicate using WP
[Brumley07]

Filter over finite domain

↓ Sound Approximation

Regular Expression Filter
(e.g., Solve WP) [Brumley06]

Fast, potentially many false negatives

Sound = No false +

Filter

58

Talk Outline

1. Automatic Patch-Based Exploit Generation

- Generate inputs that execute specific line of code (weakest precondition)
- Results



2. Automatic Input Filter Generation

- New program analysis approach to filter generation
- Filters have accuracy guarantees



3. Vine Platform for Binary Analysis

- Infrastructure



4. Questions and Answers

59

Binary Code is Everywhere

Vine targets x86

But understanding x86 semantics is challenging:

- x86 is complex
- 100's of instructions

```
add a,b  
goto L if carry
```



```
a = a+b  
parity flag = ...  
carry flag = ...  
auxiliary carry flag = ...  
zero flag = ...  
signed flag = ...  
overflow flag = ...  
... code for goto ...
```

all control flow
determined by flags

60

Vine Faithful Binary Code Analysis

- Faithful, simplified, and explicit representation of binary code

- Vine Principle: Distinguish what we know from what we don't

Vine Intermediate Language

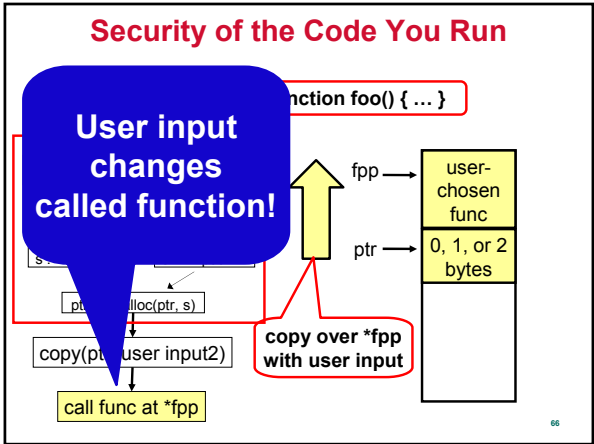
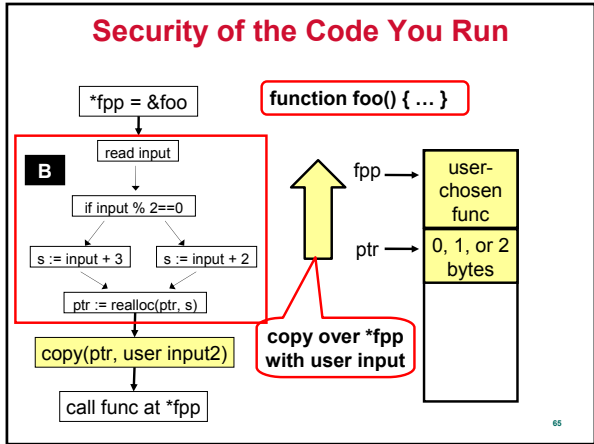
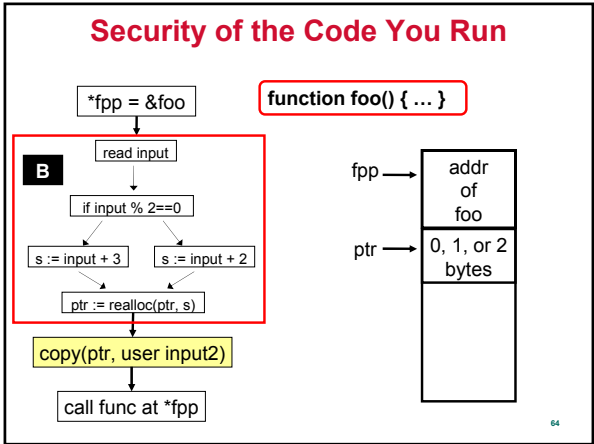
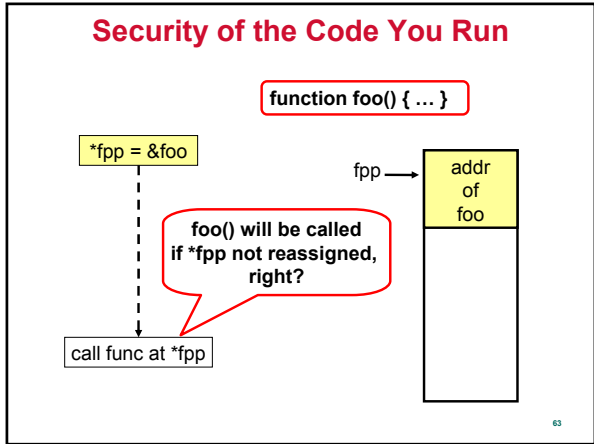
```
lval := exp  
| goto exp  
| if exp then goto exp1 else  
  exp2  
| return exp  
| call exp  
| assert exp  
| special exp  
| unknown (effects)
```

61

Security of the code you run

- Binary analysis is different than source
 - Memory instead of buffers (mem[32-bit integer] = 8-bit value)
 - Jumps to middle of "functions"
 - Few types
 - Unstructured
- Vine targets analyzing programs with potentially few abstractions
 - Problems provide assumptions, not Vine

62



Vine

Binary-Centric Approach: Vine makes it possible

Vine

- Binary code is everywhere
 - Vine addresses engineering challenges of x86
- Security of the code you run (not just the code compiled)
 - Low-level details matter
 - Vine is an infrastructure for developing and implementing binary program analyses
- Implementation:
 - 1.5 years old
 - Approx 16,500 lines C++ to raise to Vine
 - Approx 21,000 lines of OCaml for analysis & project
- 8 peer-reviewed pubs, 4 PhD Thesis, Research Partnerships with Industry (Symantec & Others) and Universities (U.Pitt & Berkeley)

67

Published Vine Projects

- **Exploit Gen** [IEEE08]
- **Filter Gen** [IEEE06,CSF07]
- **Deviation Detection** [Usenix sec07 best paper]
- **Malware analysis** [Yin et al]
- **Reverse Engineer Protocols** [Cabellero et al]
- **Etc.**

New Vine Projects

- **Secure Patch Distribution**
- **Analysis-Resistant Malware**
- **Patch Correctness** (e.g., for medical equipment, voting, etc.)
- **Etc.**

68

Research Contributions

- **Automatic Patch-Based Exploit Generation**
 - WP: Generate input that executes line of code
 - Delayed Patch Attacks are Practical
 - Current patch distribution schemes need to be redesigned
- **Automatic filter generation**
 - New approach that provides accuracy guarantees
 - Transitioning from research to practice
- **Vine: Binary analysis techniques make it possible**
 - Vine enables security-relevant program analysis
 - Fuels a variety of important research
- **Additional research I haven't told you about**
 - Net security, Source-code analysis, How I broke RSA, etc.

69

Thank You

Questions?

Contact: dbrumley@cmu.edu

70

Related Work

- **Bug Finding**
 - Fuzzing, static bug finding, dynamic bug finding, etc.
 - Does not address our scenario
 - Techniques may be complementary
- **Automatic test case generation**
 - EXE [Cadar et al]
 - Dart [Godefried et al]
 - May produce exponential size predicates, WP $O(n^2)$
- **Binary Analysis**
 - CodeSurfer/X86 [Reps & Balakrishnan 04-current]
 - Phoenix [Microsoft]
- **Filter Generation**
 - Work cited in talk
 - Bouncer/Vigilante [Costa et al SOSP05/07]
 - They use exponential algorithm, brumley07 is quadratic