

RETROFITTING SECURITY IN INPUT PARSING ROUTINES

**Jayakrishna Menon, Christophe Hauser, Yan
Shoshitaishvili, Stephen Schwab**

`{jmenon, hauser, schwab}@isi.edu`

`yans@asu.edu`

MODERN DEFENSES

- OS defenses (ASLR, DEP).
- Compiler-level defenses (e.g., stack canaries).
- Code audit tools.

VULNERABILITIES

- Many programs are still written in unsafe languages like C/C++.
- Memory corruption vulnerabilities remain prominent.

PARSERS

- Directly exposed to user input.
- Many custom implementations in unsafe languages (C/C++).
- Over 170 vulnerabilities reported in various parsing mechanisms since 1999.
- Varying semantics and the abundance of string manipulations make their implementation error-prone.

SOLUTION SPACE

DESIGN TIME SECURITY

- Parser libraries.
- Parser generators.
- Formal methods.

POST-DESIGN SECURITY

- Code audits.
- Refactoring/inserting correct parsers.
- No source code?

BINARY-LEVEL APPROACH

- Source code not always available (legacy code, uncooperative editors, untrusted IoT devices).
- What you see is not what you execute: compiler bugs, compiler “backdoors” e.g., XCodeGhost (linking malicious code into executables).



WYSINWYX

CHALLENGES

SCALING PROBLEM

Program analysis techniques are difficult to automate in a scalable and precise manner.



STATIC ANALYSIS

- Scalable.
- Imprecise.

SYMBOLIC EXECUTION

- Precise.
- Unscalable.

DYNAMIC ANALYSIS

- Precise.
- Low coverage.



SOURCE CODE

- Types.
- Variable names.
- Functions.
- ...

BINARY

- Registers.
- Memory locations.
- Basic blocks.
- ...

HOW TO SCALE TO REAL
WORLD PROGRAMS?

TEMPLATE-BASED APPROACH

... to discover vulnerabilities based on templates corresponding to common classes of security bugs.

... to retrofit security by patching programs at the binary-level.

INITIAL APPROACH

- Focuses on overflows in buffers allocated statically on the stack.
- template-based:
categorize causes of vulnerabilities into three classes.
- Combines static analysis and symbolic execution.

CLASSES/TEMPLATES

- Unconstrained input.
- Under-constrained input size.
- Unchecked termination condition.
- ...

UNCONSTRAINED INPUT.

Improper usage of functions that do not check for sizes such as strcpy, sprintf etc.

EXAMPLE 1: CVE-2003-0390



```
int opt_atoi( char *s) {  
    char buf[1024];  
    char *fmt = "String [%s] is not valid";  
    sprintf(buf, fmt, s);  
}
```

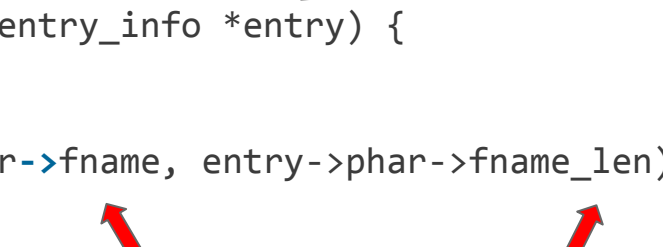


UNDER-CONSTRAINED INPUT SIZE.

Improper validation of size field
in functions such as memcpy.

EXAMPLE 2: CVE-2015-3329

```
void phar_set_inode( phar_entry_info *entry) {  
    char tmp[1024];  
    memcpy(tmp, entry->phar->fname, entry->phar->fname_len);  
}
```



UNCHECKED
TERMINATION
CONDITION.

Performing operations on
(possibly) incorrectly terminated
strings.

2-STEP ANALYSIS APPROACH

Static analysis

CFG

DDG

*Identify string
manipulation
functions.*

*Identify user
input.*

*Identify
destination
buffers (sinks).*

*Analyze backward
data-dependency.*

*Dangerous
program paths.*

Symbolic analysis

SE

*Path
constraints.*

(MEMORY CORRUPTION CAUSED BY UNSAFE BUFFER MANIPULATION)

ANALYSIS RESULTS

	Static Analysis	Symbolic execution	Overall
False positive rate	6.6%	0%	0% *
False negative rate	40%	0% *	40%
Time	1-260s	1-400s	2-660s

NEW BUGS

2 new bugs found in the binary code of common opensource projects and libraries (in a semi-automatic setting)

RETROFITTING SECURITY: BINARY PATCHING

ADDING THE MISSING CHECKS

- Remember: we focus on stack buffers.
- On the identified program paths, we constrain the user input such that:

```
user_input_size <  
stack_buffer_size
```


ADDING THE MISSING CHECKS

When the constraints are violated, we crash the program.

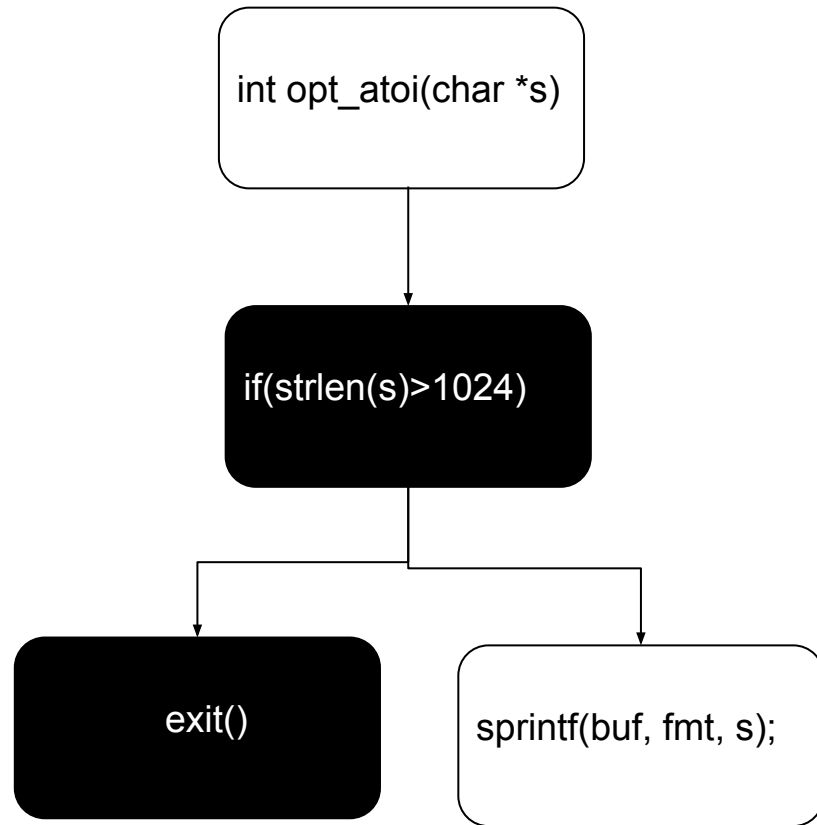
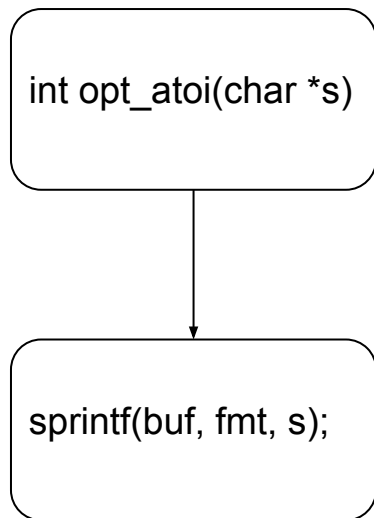
This is equivalent to e.g.,
`__sprintf_chk()`

PATCHING THE BINARY

Static reassembly problems:
breaking internal program
references.

Partial solution: inject
trampoline gadgets in padding
bytes between functions (up to 15
consecutive NOPs).

INSERTING CHECKS



MORE TEMPLATES

NEW TEMPLATE

Memory allocation errors

- ... authentication errors.
- ... misuses of cryptographic APIs.
- ... information leakage.

NEW BUGS

12 new bugs found in the binary code of common opensource programs and libraries (in a fully automated setting).

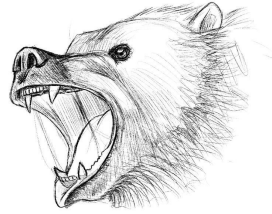
DISCUSSION

Lightweight and scalable approach.

... but high rate of false negatives.

... limited patching capabilities.

STUMBLING BLOCKS



Data structure
recovery.



Pointer
aliasing.

FUTURE WORK

- Improve data dependence tracking.
- Leverage static reassembly techniques.
- More vulnerability templates.
- Apply to large corpus of IoT firmware.

KEY TAKEAWAYS

- Templates per vulnerability class.
- Scalable, two-level approach based on a combination of static analysis + symbolic execution.
- High-precision: we can infer semantic-agnostic patches for each class.
- New bugs.

?