Practical Techniques for Regeneration and Immunization of COTS Applications

Lixin Li
Mark R.Cornwell
E.Hultman
James E. Just

Global InfoTek, Inc

R. Sekar

Stony Brook University

(Research supported by DARPA, NSF and ONR)

Problem

- Windows of vulnerability in a production system
  - Zero-day vulnerability is a serious concern
  - Exploits produce faster than patches
  - End users take time to deploy patches
- Ideal solution requires:
  - Defends against most common attacks
  - Protects application integrity as well as availability
  - Low performance overhead
  - Working on COTS

Related Work

- Taint-tracking:
  - Drawbacks
    - Intrusive instrumentation and
    - High overhead and
    - Somewhat language-specific
- Automatic Signature Generation:
  - Drawbacks
    - Heavy weight or
    - Requires accurate attack replay

GITI Global InfoTek, Inc.

7/7/2009

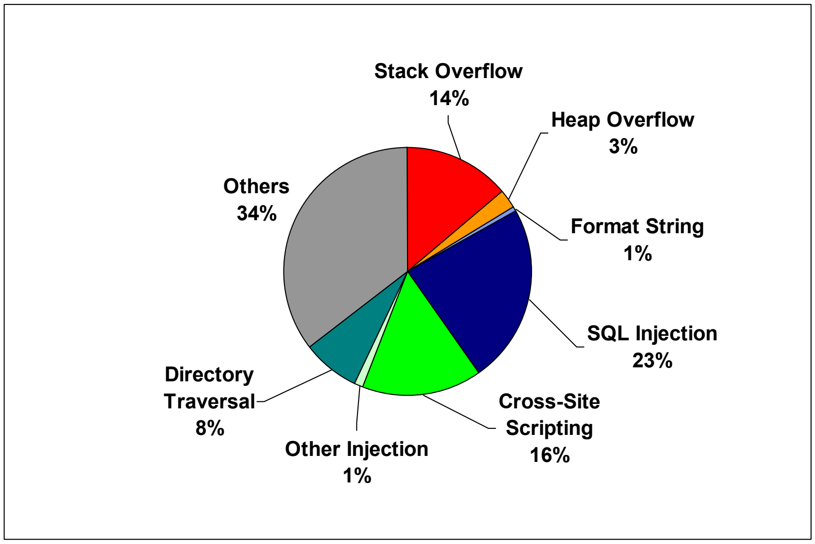
Our Research

- Approach applicable to most popular attacks, from memory corruption attacks to string injection attacks
- Our project, RAMSES (Regeneration and Immunization Services), provides regeneration and immunization services
  - Regeneration provided by our ASR implementation on Windows binary (implemented prior to Windows Vista)
  - Immunizes systems against zero-day attacks, preserve integrity and availability
    - Inspired by biological immune system to learn from attacks
    - Provides much targeted and effective response without damaging host
- Protect against attack variants and brute-force attacks
  - Vulnerability-oriented signature generation

GITI Global InfoTek, Inc.

7/7/2009

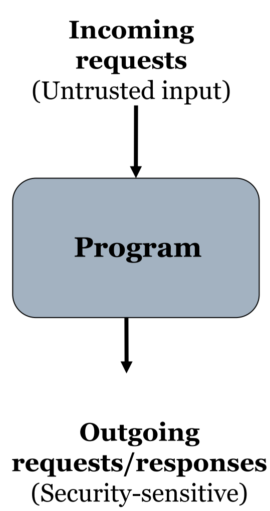
Attack Space of Interest (CVE 2008-09)



7/7/2009

Attack Scenarios

- String injection vs Memory corruption attacks
  - String injection attacks:
    - Target output requests to access protected services (SQL database), subsystems (command interpreters)
  - Memory corruption attacks:
    - Target critical data structures (stack, heap,…)
    - Output interface is not clearly defined
    - Attacker intends to hijack execution to run attacker provided arbitrary code
- Attack: use maliciously crafted input to exert unintended control over output operations
- Detect "exertion of control"
  - Output depends on input (taint)
- Detect the degree of intended control is necessary for string injection.



7/7/2009

Approach Overview

- Taint inference: an *efficient* and *non-invasive* alternative to taint analysis
  - Analyze observed inputs and outputs
  - Standard library API Interception is mostly needed
  - Language-neutral
- Attack detection: Use taint-inference to correlate attack with causative input
  - Use Address-Space Randomization (ASR) to detect memory corruption attacks
    - Use memory analysis on corrupted areas for attacks detected by ASR
  - Use syntax and taint-aware policies to detect string injections
    - Leverage interplay between taint and structural changes to output requests
- Immunization: filter out future attack instances
  - Input Filter
  - Output Filter

7/7/2009

GITI Global InfoTek, Inc.

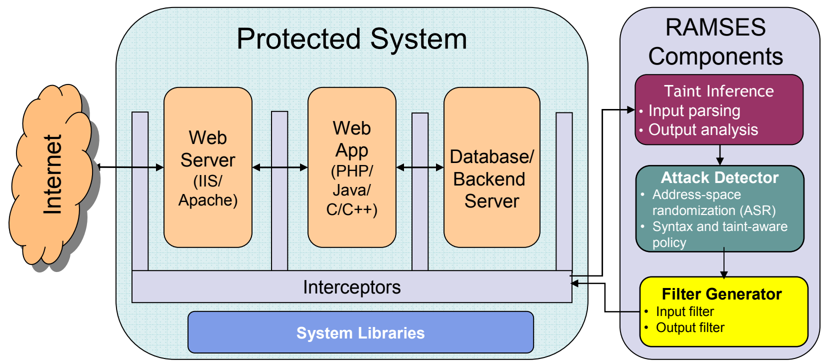
Taint Inference

- Infer taint flow by observing inputs and outputs
  - Assumption:
    - No arbitrary transformation of Input
      - Standard transformation like encoding can be accommodated
- Key observation:
  - *Major* parts of input appear at output with *minor* changes
    - Vulnerable buffer corrupted after a buffer overflow and before the attack is detected (taint "untainted")
    - Simple transformations are common in web applications
- Solution: use approximate substring matching
  - Given an input *I* and an output *O*, substring *o* of output *O* contains data from input *I* if there is an approximate string match between *I* and substring *o* (the edit-distance between *I* and *o* is less than a given threshold)
- Standard approximate substring matching algorithms have quadratic time and space complexity
- Our fast approximate substring match algorithm has linear-time complexity

7/7/2009

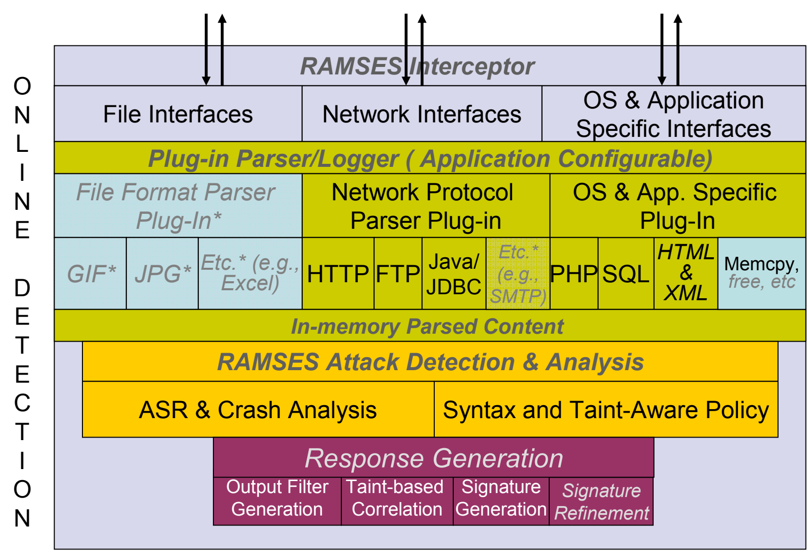
GITI Global InfoTek, Inc.

System Components



7/7/2009

System Architecture



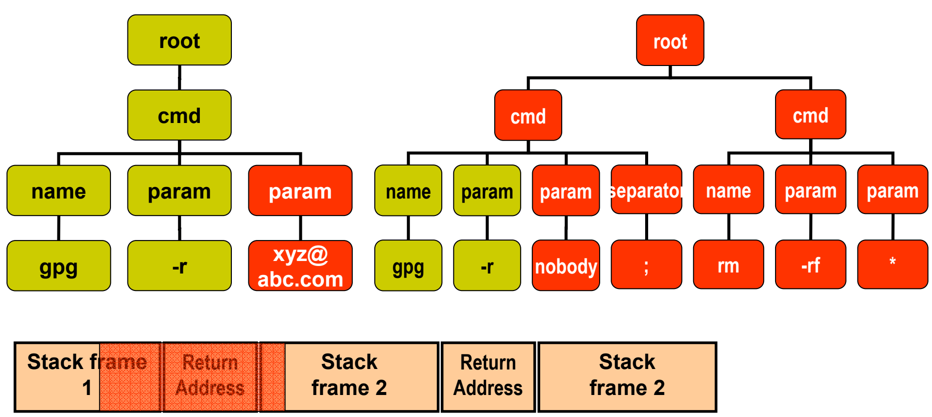
7/7/2009

Lexical and Syntactic Confinement

- Basis of an uniformed approach to detect both memory corruption attacks and string injection attacks
  - Key observation:
    - Attack-bearing input breaks output structures
- String injection attacks
  - Lexical confinement
    - Tainted data should not flow past the end of tokens in output parse tree
  - Syntactic confinement
    - Tainted data should not straddle two sub-trees of the output parse tree
- Memory corruption attacks
  - Lexical/Syntactic confinement generally applicable
  - Buffer overflow breaks local array confinement and break stack/heap structures

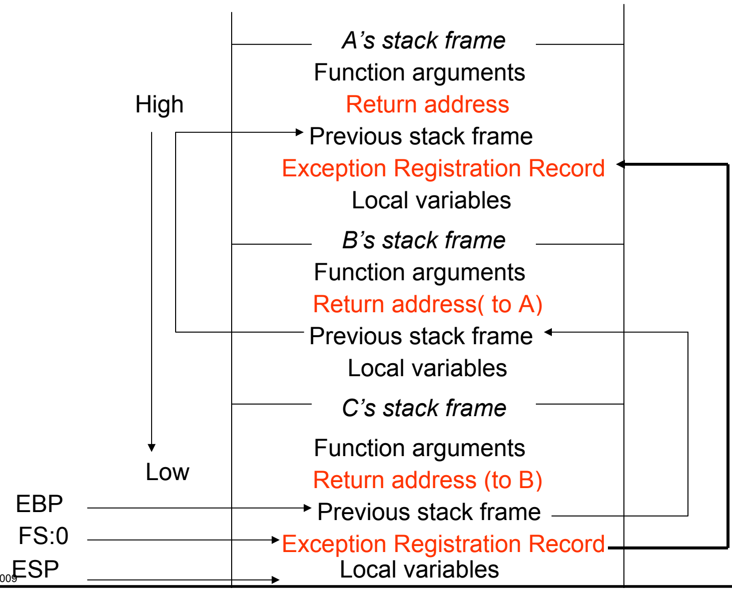
Illustration of Lexical/Syntactic Confinement

Benign Input: xyz@abc.com **Attack Input: nobody; rm –rf ***



Stack Frame Structure

- Windows Stack Structure



Heap Block Structure

Size		Previous Size	
Segment Index	Flags	Unused	Tag Index
FLink			
BLink			

Windows Free Heap Block Structure

Memory Security Analysis Steps

- Input Parsing:
  - Parsing input into meaningful (field, value) pairs
    - To simplify taint-inference
    - Provide semantic context for signature generation
- Crash exception analysis
  - Corruption Area and Targets Analysis
  - Buffer Confinement Analysis
    - Taint inference
    - Approximate string match naturally handles “untaint” after buffer overflow
- Signature generation
  - Length-based
  - Apply at the right semantic context

Memory Security Analysis & Implementation

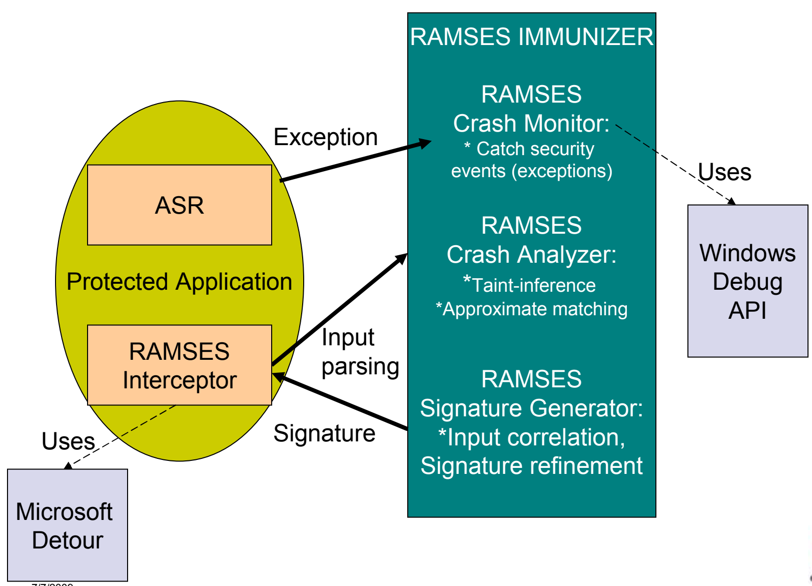
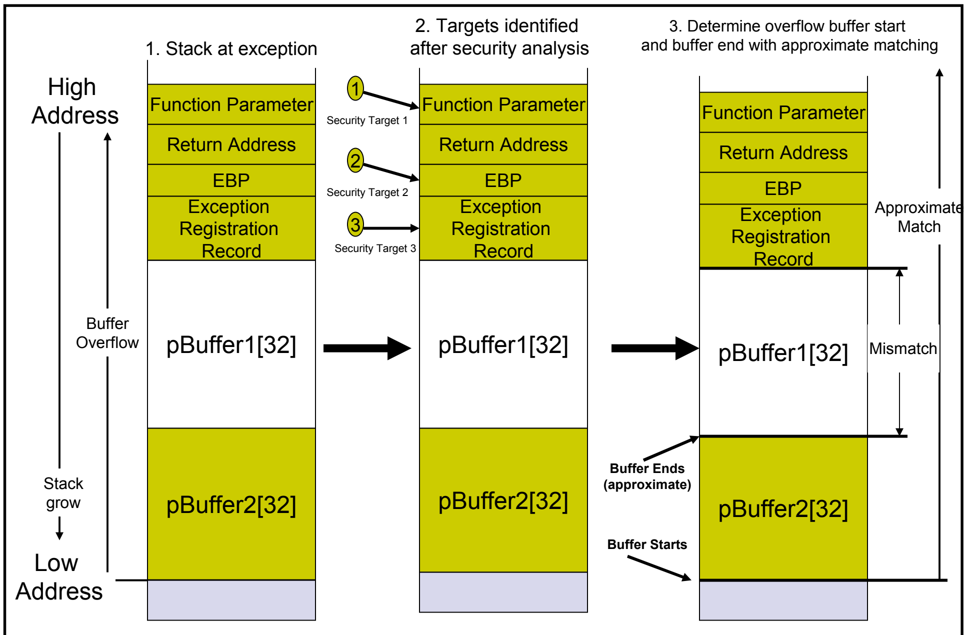


Illustration of taint infer a stack buffer overflow

1. pBuffer2[32] is a vulnerable buffer, input overflow pBuffer2,another local buffer pBuffer1[32], Exception Handler, Saved EBP, return address and beyond.
   - pBuffer1[32] is “untainted” before attack is detected
2. Crash security analysis identified corrupted area and interesting targets
3. Apply approximate string match on input value from input parsing step and corrupted area to identify starting boundary and ending boundary of the vulnerable buffer.
   - Despite the “gap” caused by “untaint”, approximate string match succeeds.

Global InfoTek, Inc.

7/7/2009



Global InfoTek, Inc.

7/7/2009

Filters

- Input filters block attack input from passing through input functions
  - Return error code for the input function
  - Servers written to expect network errors
- Input filter is used for memory corruption attacks
  - Output interface is not clearly defined
  - Output filter is usually too late
    - Output is usually in the form of program crash
- Output filter is used for string injection attacks
  - Block policy-violating outputs from carrying out output functions
  - Intercept the output function and returning an error code
  - Recovery is usually possible for string injection attacks

Global InfoTek, Inc.

7/7/2009

Preliminary Results (cont)

- Input filter: Memory corruption attacks

Vulnerability Type	Attack Target	Payload Type	Attack Description	Vulnerable Buffer Identified	Blocking Signature Generated
Synthetic	MEVS	Working exploit	Stack buffer overflow to overwrite return address	Yes	Yes
Synthetic	MEVS	DoS	Stack buffer overflow to overwrite return address	Yes	Yes
Synthetic	MEVS	Working exploit	Stack buffer overflow to overwrite SEH	Yes	Yes
Synthetic	IIS ISAPI extension DLL	Working exploit with payload encoded	Stack buffer overflow to overwrite SEH	Yes	Yes
Synthetic	MEVS	Working exploit	Heap Overflow Freelist[00] to trigger double pointer unlink	Yes	Yes
Synthetic	MEVS	DoS	Heap Overflow [1 - 127] to trigger double pointer unlink	Yes	Yes
Synthetic	MEVS	Working exploit	Heap Overflow Lookaside list to trigger double pointer unlink	Yes	Yes
Synthetic	MEVS	DoS	Heap Overflow triggering blocks coalesce and double pointer unlink	Yes	Yes
Synthetic	MEVS	DoS	Heap overflow with original input reversed before overflow	No	No
Real world (CVE-2004-1134)	IIS5	Working exploit	Overflow a stack buffer in w3who.dll to overwrite SEH	Yes	Yes
Real world (OSVDB-20909)	FreeFTPd 1.08	Working exploit	Overflow a stack buffer in freeFTPd service to overwrite SEH	Yes	Yes

7/7/2009

Preliminary Results

- Output filter: Web Applications

Application	Language	Size (lines)	Environment	Attacks	Comments	Detection	False Positive
phpBB 2.05	PHP/C	34K	IIS, Apache	SQL injection	CAN-2003-0486	Yes	None
SquirrelMail 1.4.0	PHP/C	42K	IIS, Apache	Shell cmd injection	CAN-2003-0990	Yes	None
SquirrelMail 1.2.10	PHP/C	35K	IIS, Apache	XSS	CAN-2002-1341	Yes	None
PHP/ XMLRPC	PHP/C	2K	IIS, Apache	PHP cmd injection	CAN-2005-1921	Yes	None
AMNESIA(5 apps)	Java/C	30K	Apache/ Tomcat	SQL injection	21K attacks, 3.8K legitimate	100%	0%
WebGoat	Java		Tomcat	HTTP response splitting Shell cmd injection		Yes Yes	None None

7/7/2009

End-to-end Performance Overhead

- Memory signature generation time only incurs when exception happens, usually less than 250 ms cpu time for stack overflow and less than 800 ms for heap overflow
- Web Application Performance Overhead

Application	Size (LOC)	# of Requests	Response time (sec)	Overhead
Bookstore	9552	605	20.7	1.7%
Empldir	3028	660	17.3	3.4%
Portal	8775	1080	31.7	5.1%
Classifieds	5726	576	18.0	4.3%
Events	3805	900	23.0	3.1%
Total	30886	3821	110.7	**3.5%**

7/7/2009

Contact

- Lixin Li, Principal Scientist
  Global InfoTek, Inc., Reston, VA
  - nli@globalinfotek.com

- James E. Just, Director of Research
  Global InfoTek, Inc., Reston, VA
  - jjust@globalinfotek.com

- R. Sekar, Professor of Univ. Stony Brook
  - sekar@cs.stonybrook.edu

7/7/2009

Questions?

7/7/2009