Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Namespaces

Outline

- Sects 3.4-3.6
- Unix File Races (Exploits)
- Unix File Races (Defense)

Detecting Buffer Overruns

- Static analysis tool to detect buffer-overrun vulnerabilities in C source code
  - Build ICFG
  - Collect constraints suitable for a linear program solver
  - Solve the constraints
  - Find bugs

Detecting Buffer Overruns

- Static analysis tool to detect buffer-overrun vulnerabilities in C source code
  - Build ICFG
  - Collect constraints suitable for a linear program solver
  - Solve the constraints
  - Find bugs

Detecting Buffer Overruns

- Static analysis tool to detect buffer-overrun vulnerabilities in C source code
  - Build ICFG
  - Collect constraints suitable for a linear program solver
    - Compute constraints with flow-insensitive and context-insensitive approach
    - Remove constraints that trouble the linear program solver – are infeasible or unbounded
  - Solve the constraints
  - Find bugs

Detecting Buffer Overruns

- Static analysis tool to detect buffer-overrun vulnerabilities in C source code
  - Build ICFG
  - Collect constraints suitable for a linear program solver
    - Compute constraints with flow-insensitive and context-sensitive approach
    - Remove constraints that trouble the linear program solver – are infeasible or unbounded
  - Solve the constraints
  - Find bugs

Context Insensitivity

- At each call-site
- Assign the actual-in vars to the formal-in vars
- Assign the formal-out to the actual-out
- See Figure 3.3
  - *buffer* is bound by *buf* (and *header*)
  - *cc2* is bound by return of *copy_buffer*
- *cc1* and *cc2* get the same values
  - Does that seem reasonable?

Constraint Inlining

- Like inlining functions
  - What is that?
- Create a fresh constraints for the called function at each call site
  - Use unique versions of the local and formal vars for each call site
  - I.e., actual-in assigned to renamed formal-in
  - I.e., renamed formal-out are assigned to actual-out
  - What is the result for analysis?

Constraint Inlining Issues

- Doesn't work for recursive function calls
- The number of constraint vars may be exponentially larger than the number of context-insensitive constraints
- What can we do?

Summary Constraints

- Goal: Eliminate constraints based on local variables
  - Call remaining *summary constraints*
- Use only formal parameters and globals
  - See Fig 3.10
- Variable elimination techniques are known

Fourier-Motzkin Elimination

- Input
  - Set of constraints *C* and set of variables *V*
  - Variables are formal and globals to be retained
- Iteratively eliminates variables not in *V*
  - *copy!alloc!max >= buffer!used!max – 1*
  - *copy_buffer!return!alloc!max >= copy!alloc!max*
- Becomes
  - *copy_buffer!return!alloc!max >= buffer!used!max – 1*

Fourier-Motzkin Elimination

- Not always that easy in general, however
  - To eliminate *v*, where *m* constraints use *v* and *n* constraints define *v*
  - Requires *m* * *n* constraints
- Because buffer overflow constraints are *difference constraints*, we can be more efficient
  - Reduces to all-pairs shortest/longest path

Fourier-Motzkin Elimination

- Consider a function that does not call other functions or only calls functions with summaries
- To produce summary constraints C in terms of variables V construct a graph for constraints in C
  - Vertices are constraint variables in C
  - Edges for relationships in constraints
    - *v1* >= *v2* + *w* results in an edge from *v2* to *v1* of weight *w*
  - Find longest path between any two variables in *V*
    - Which is two for the example

Now for Context-Sensitivity

- Build constraints between function variables and formal parameters through above method
  - Figure 3.12
- Find relationship between cc2 and formal parameters using DAG

Results

PENNSTATE

Program	LOC	Warnings	Errors
wu-ftpd-2.6.2	18K	178	14
wu-ftpd-2.5.0	16K	139	Confirmed errors
sendmail-8.7.6	38K	295	>2
sendmail-8.11.6	68K	453	Confirmed errors
Talk daemon	900	4	0
Telnet daemon	9400	40	>1

Specific Results

- Good
  - Wu-ftpd: track relationship between pointers and buffers accurately enough
    - Track user input
  - Telnet: found a violating use of a supposedly safe function: strncpy
  - Sendmail: find failed conditional checks that cause overflow
- Less Good
  - Wu-ftpd: False positive do to lack of flow-sensitivity
  - Talk: all warnings were false alarms (although due to system)

Performance

	Wu-ftpd-2.6.2	Sendmail-8.7.6
Codesurfer	12.54s	30.09s
Generator	74.88s	266.39s
Taint	9.32s	28.66s
LP Solve	3.81s	13.10s
Hier Solve	10.08s	25.82s

- Constraints
  - Pre-taint: 22K and 104K, respectively
  - Post-taint: 15K and 24K, respectively

Context Sensitivity Impact

- Number of range variables that were refined
  - Wu-ftpd: for 7310 vars, 72 were made more precise
    - For a 1% increase in constraints
  - Compared to a 5.8x increase for constraints for inlining
    - However, inlining is more precise
    - Why?

Pointer Analysis

- Remove false negatives by handling dereferencing
  - Although not aliasing in general
- Sendmail
  - 251 warnings with pointer analysis off (295 when on)
- Tough problem

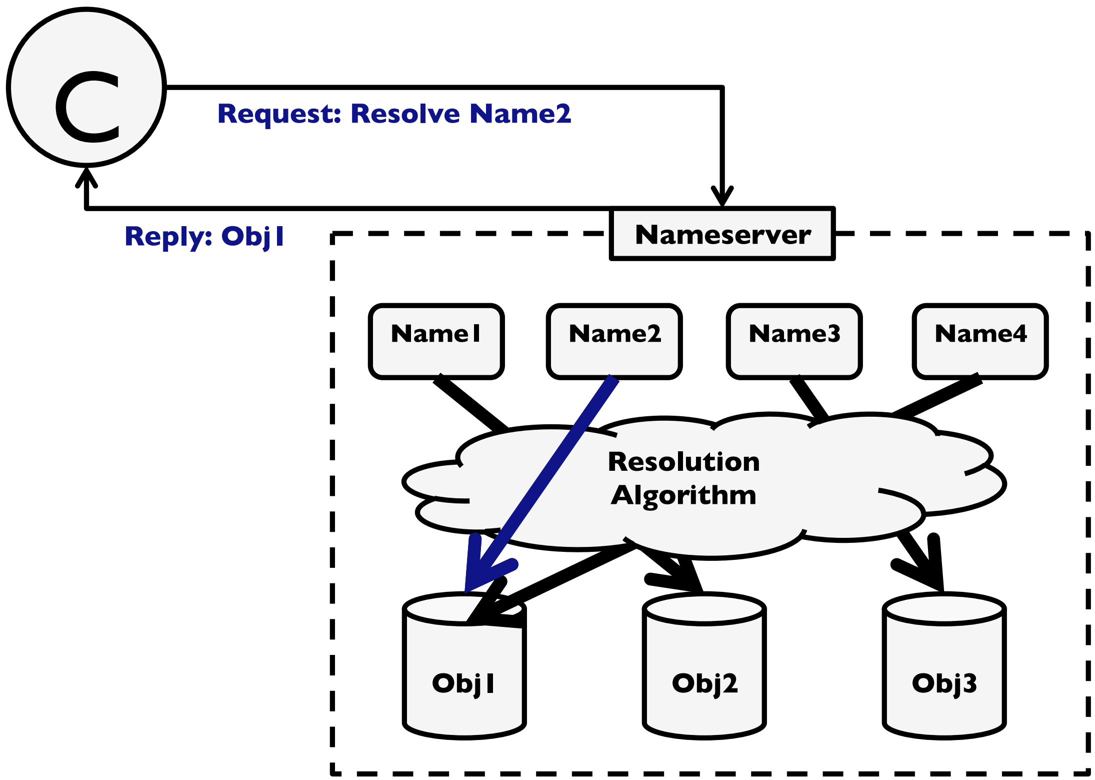
Shortcomings

- Flow-insensitivity
  - Creates false positives
  - Can use slicing to help identify
  - But, manual process to remove false positives
  - Solution: use SSA approach – lots of constraint vars
- Pointers to buffers
  - Creates false negatives
  - Because pointer analysis algorithms are flow- and context-insensitive
  - Need better algorithms – but costs time

Namespaces

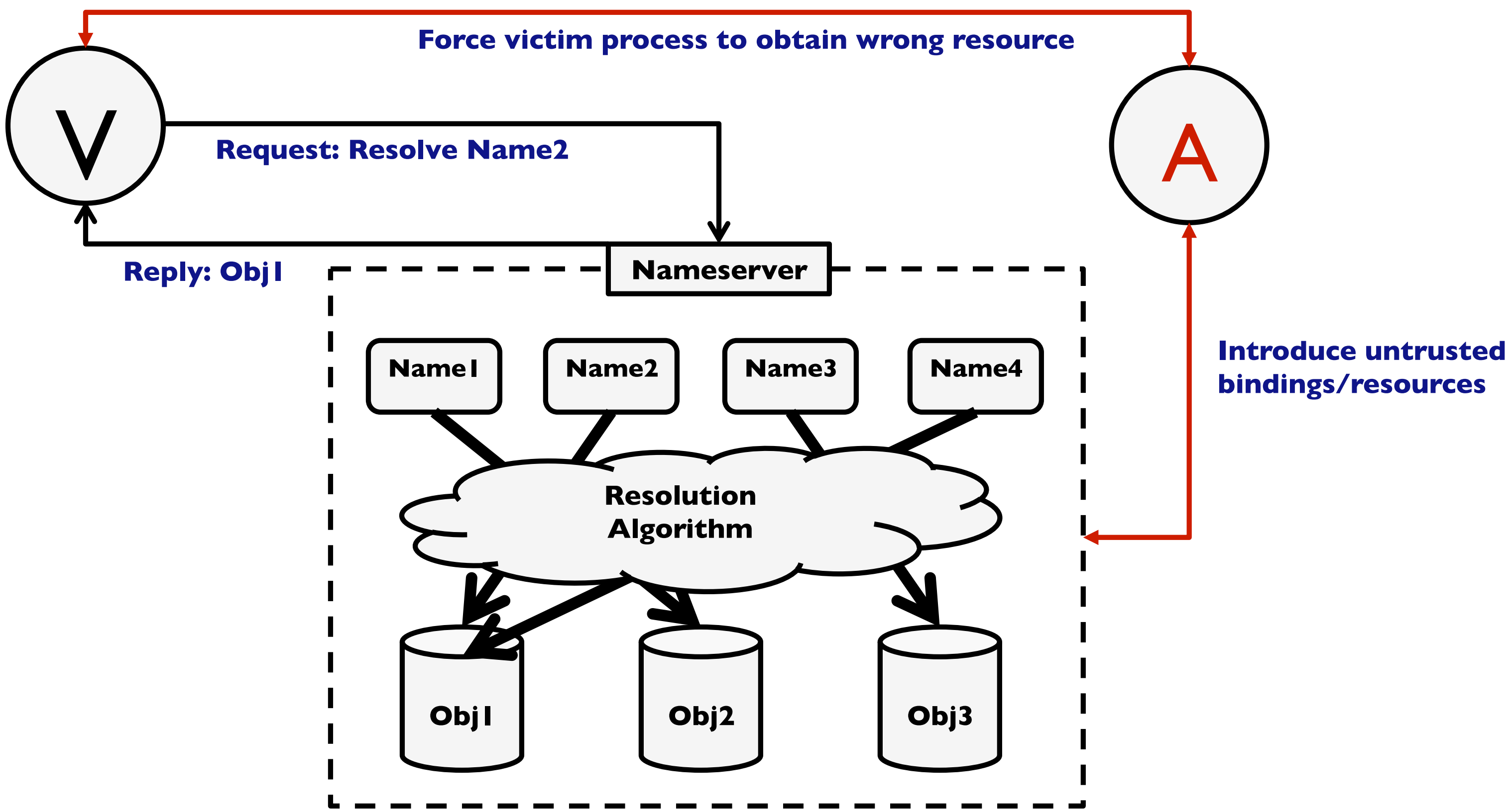
- Fundamental system mechanism
  - Simply resolves a name to an object reference for use
  - F(space, name) → reference
- Namespaces are everywhere
  - Filesystems, Domain Name Service
  - D-Bus, Android – future: cloud computing
- What kinds of problems can occur?

Name Resolution



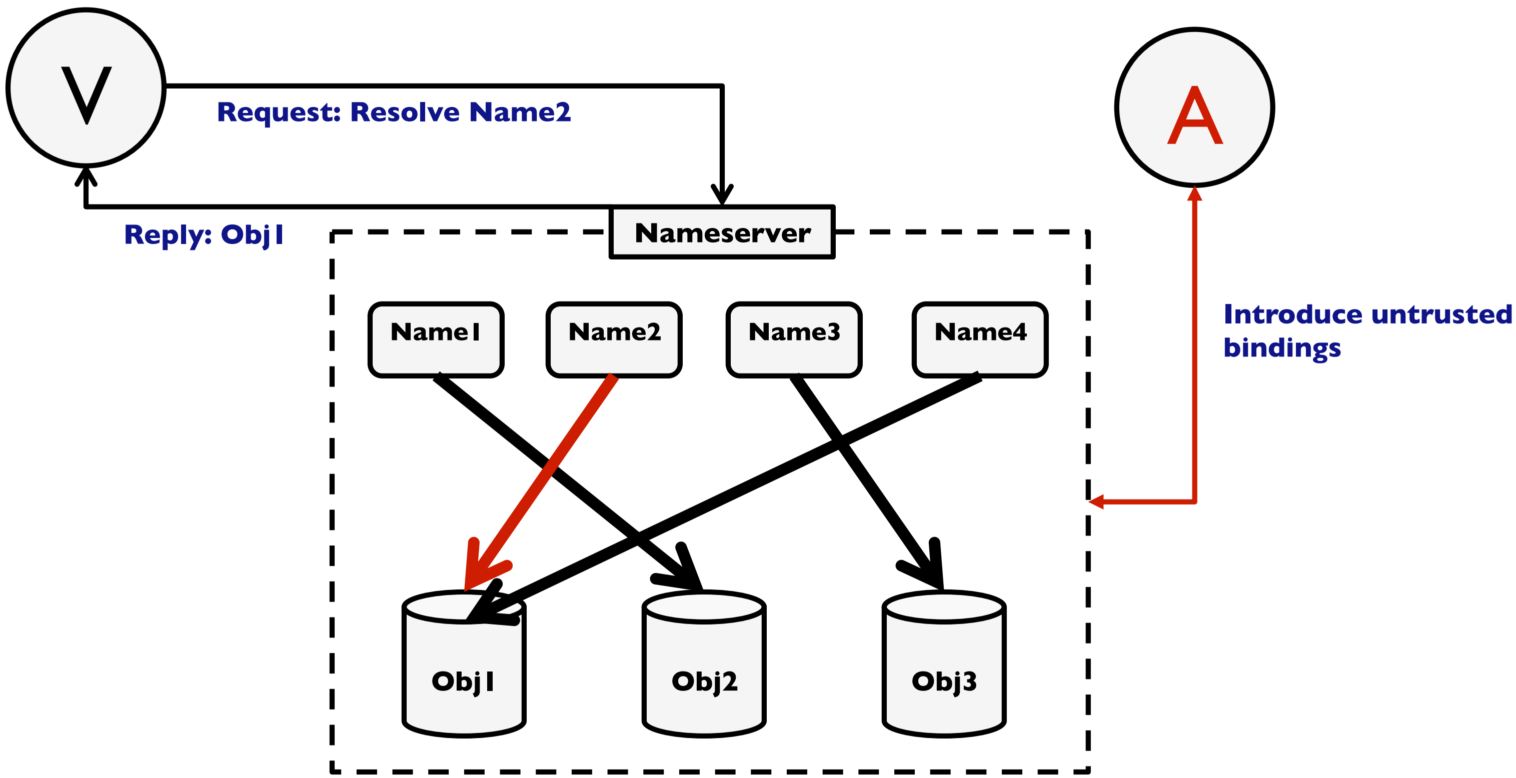
Threat Model

- Victim process and adversary process
- Adversary uses any permissions it has to try to affect name resolution



Untrusted Bindings – Pre-Binding

- Adversary pre-creates bindings that victim follows
  - Prerequisite: Predictable names



Pre-Binding Example

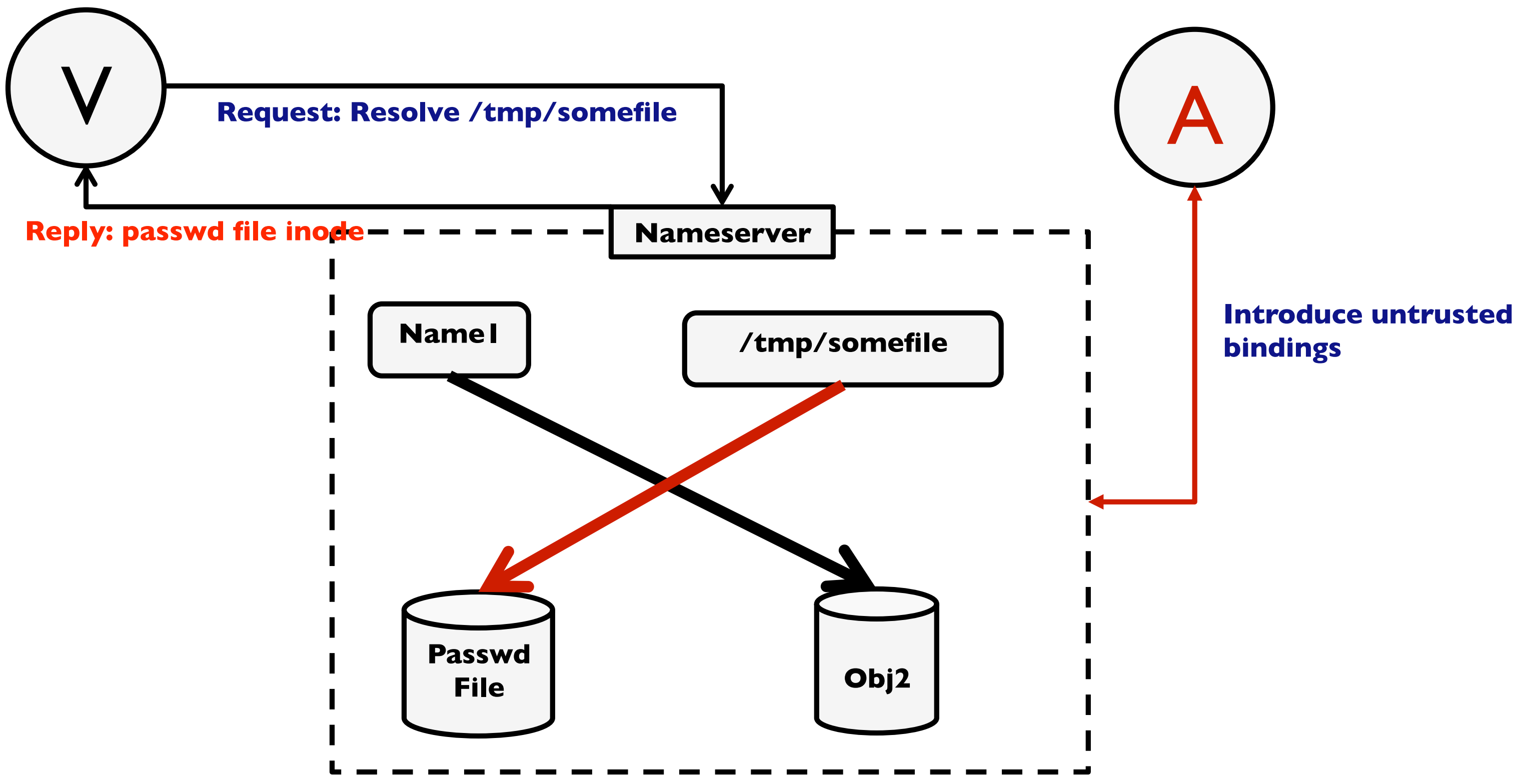
- Bash script predictable temporary file

Victim:

script.sh:

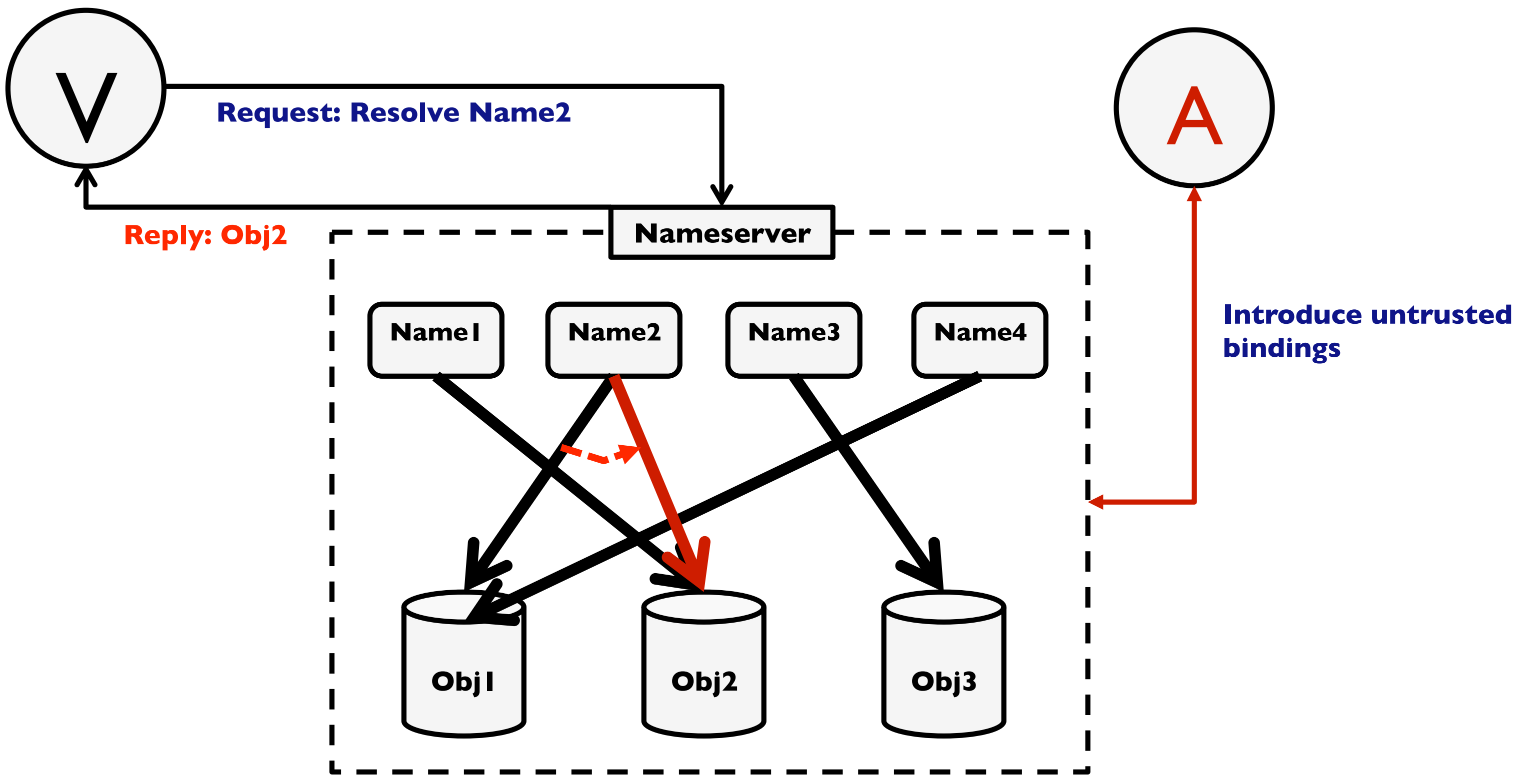
…
echo $tmpstate > /tmp/somefile

Adversary:
/* Link /tmp/somefile to point to /etc/passwd */
ln -s /etc/passwd /tmp/somefile



Untrusted Bindings - Re-binding

- Adversary modifies an already existing binding



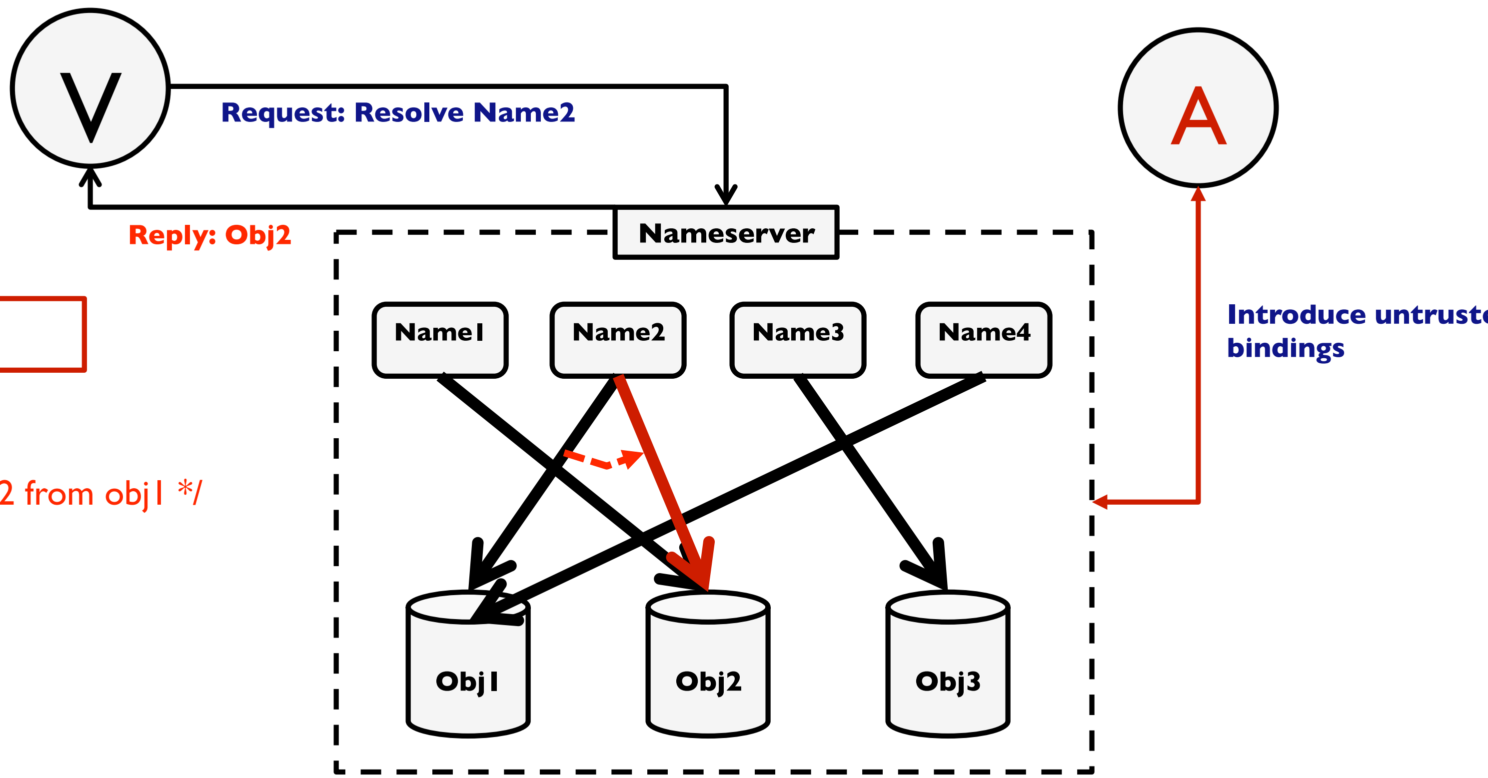
Re-Binding Example

- Linux filesystem namespace
    - Time-of-check-to-time-of-use (TOCTTOU) attack

Victim:
obj_stat = stat("name2");
/* Check obj_stat properties */
/* open obj */
obj = open("name2");

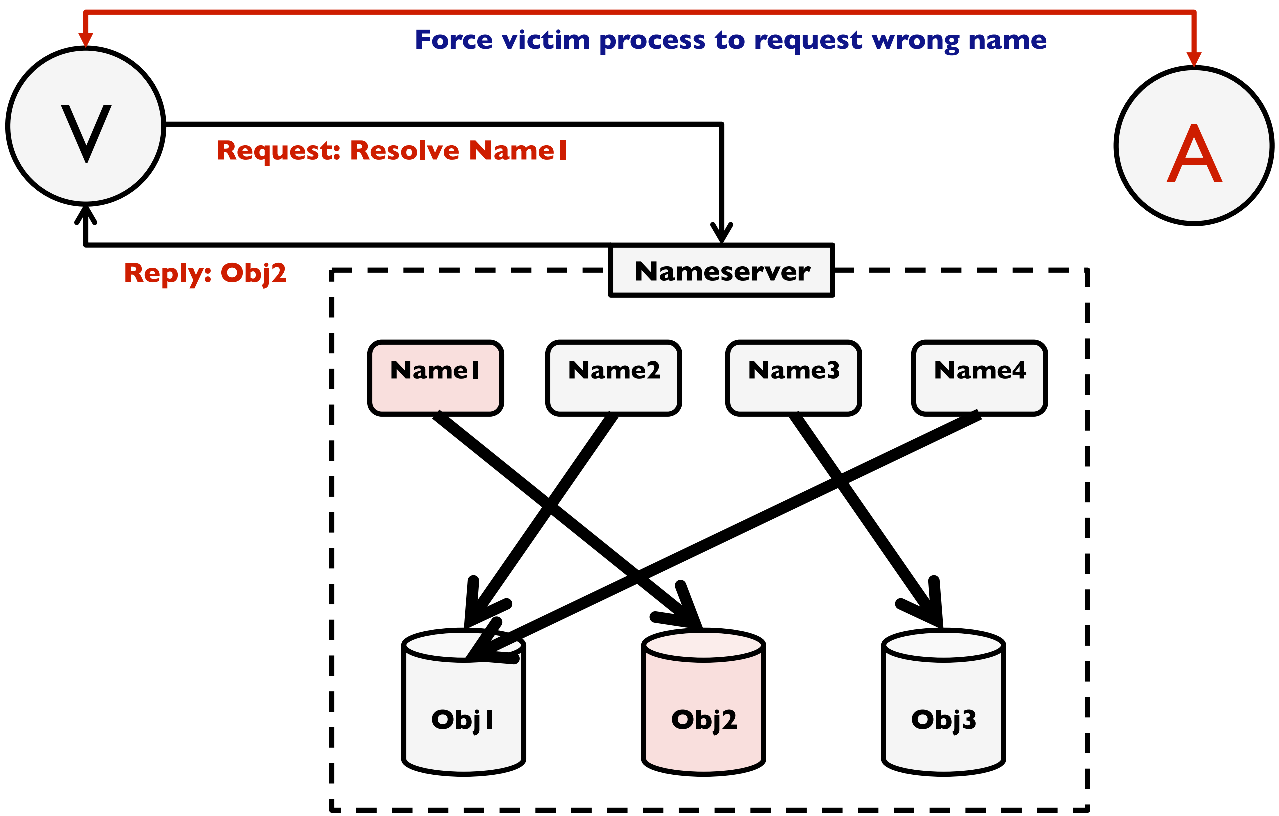
obj_stat != obj

Adversary:
/* Change name2 to point to obj2 from obj1 */



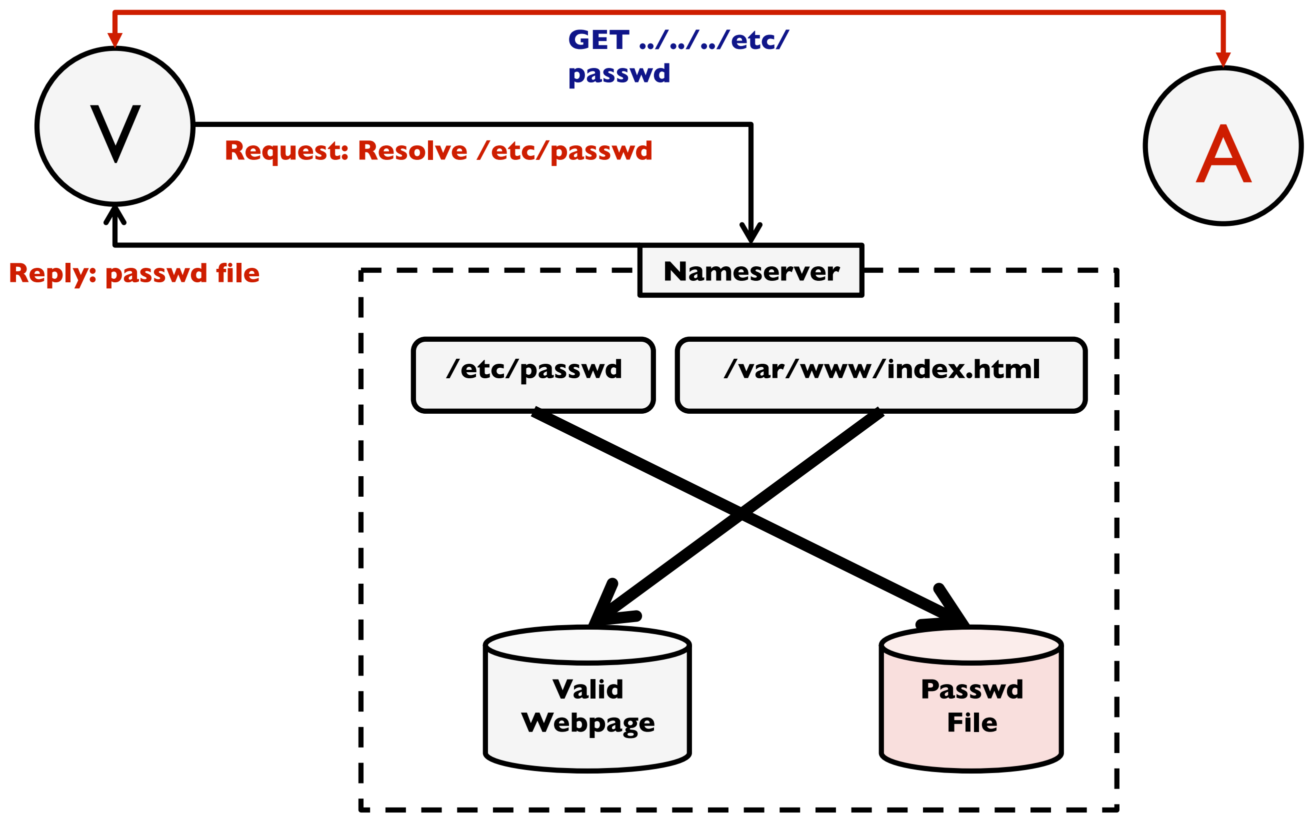
Improper Name Attack

- Adversary forces victim process to request an improper name
  - Usually due to a bug in the program



Improper Name Example

- Directory Traversal Attack
  - V is a web/FTP server

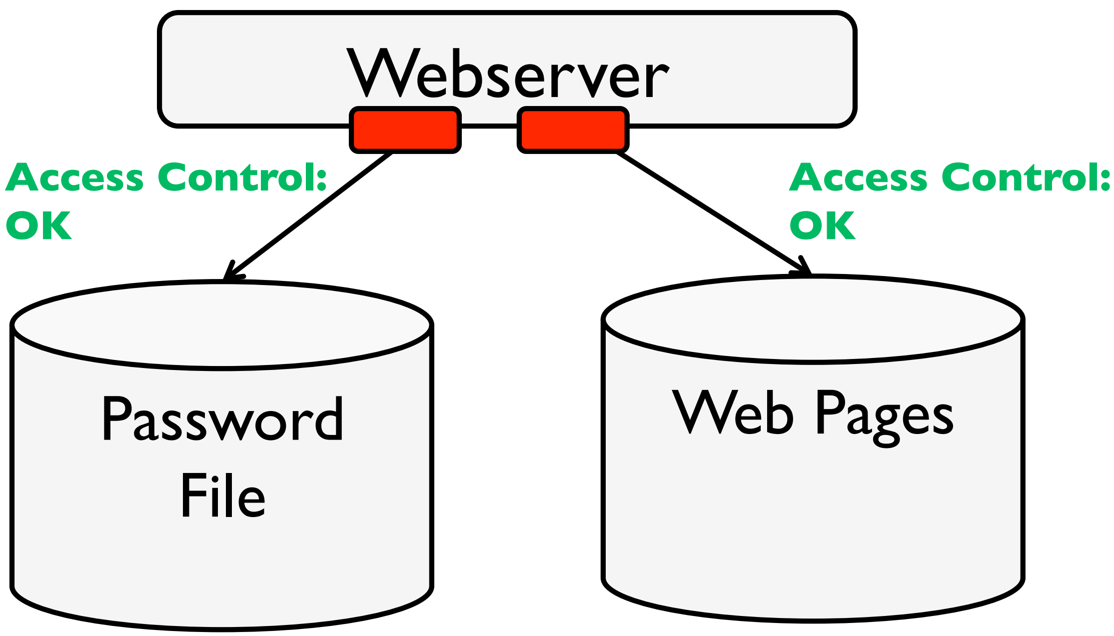


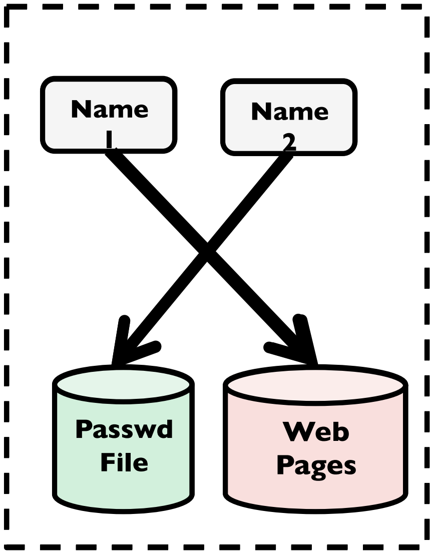
Access Control is Insufficient

- Traditional access control is insufficient to solve the problem
  - Takes into account subject, object and operation requested by subject on the object
- However, different name resolutions valid in different contexts for a single subject

Access Control Is Insufficient

- Webserver vulnerable to directory traversal
- Therefore, namespace resolution enforcement needs additional context than traditional access control
  - In this case, interface in the webserver making the call





Questions

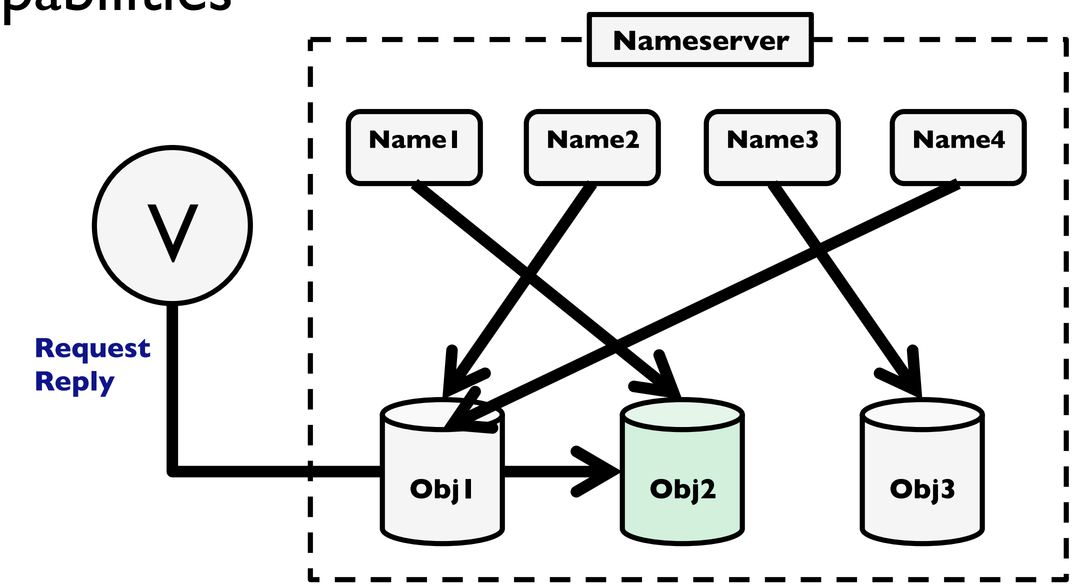
- Generic defense against namespace attacks
  - What is a generic defense?
  - Where to implement?

Existing Program Defenses

- Program API to convey intended context to OS
  - E.g.,
    - O_EXCL flag in open(): if a binding already exists, fail
    - mkstemp creates an unpredictable name
- Programmers do not always use APIs properly
  - TOCTTOU attacks first published by Bishop et al. [1996]
  - Buffer overflows known for decades
- Other bugs in programs allow circumvention
- Hence, we propose a system-level solution for namespace problems

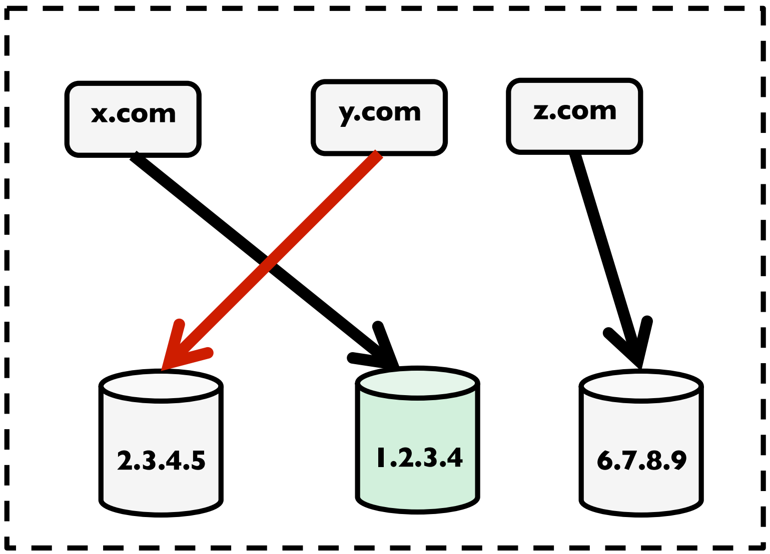
Capabilities

- Give process a capability to access a resource
- Bypass namespace completely
- Limitations
  - Resolution has to be done at some stage to get capabilities
  - Developers find indirection convenient
  - Programmers choose capabilities



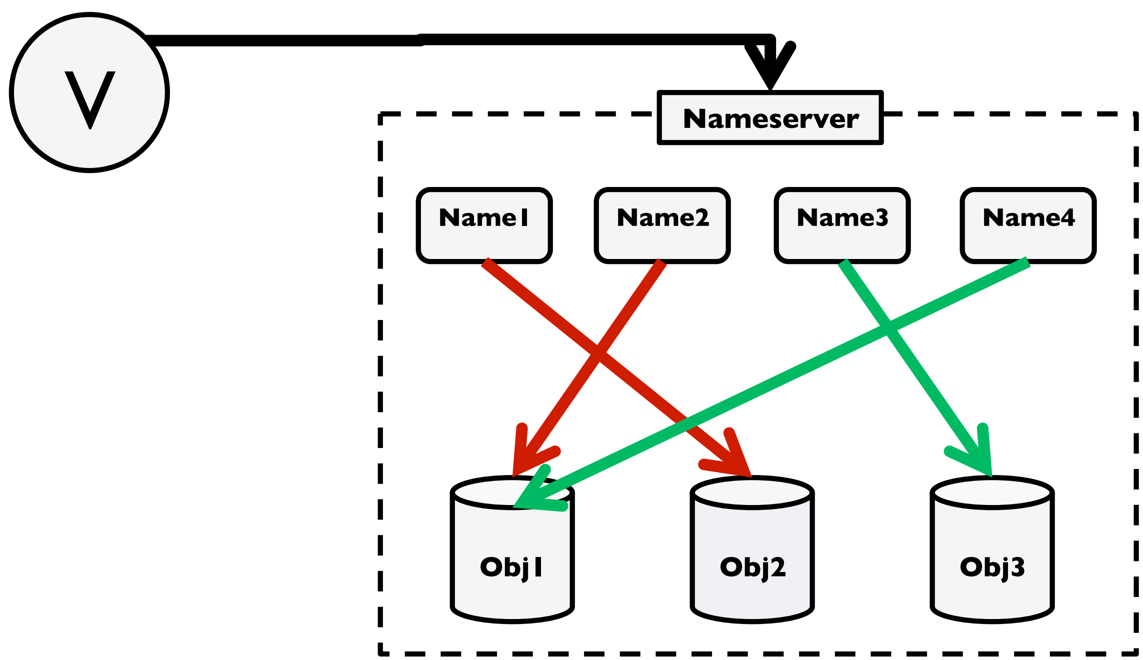
Firewalls

- Restriction on the resource fetched (by resource ID)
- Traditional Example: Network Firewalls
- IP addresses (resources) that can be accessed is limited, even if namespace (DNS) is compromised by adversarial bindings
  - E.g., pharming, locally changing hosts file
- Limitations
  - Policy manually specified
  - Applies to network only
  - Fake IP addresses



Namespace Management

- Restrict introduction of bindings to only trusted entities
- Example: Private namespaces
  - Used by container virtualization to isolate VMs (LXC, OpenVZ)
- Limitations
  - In some cases, retrieving low-integrity objects through low-integrity bindings is necessary for functionality



Namespace Management

- In recent work, Chari et al. [2010] introduce heuristics for traversing bindings in a Linux filesystem
  - Only trusted bindings (created by the same user or root) should be traversed
  - More complex heuristics for untrusted bindings
- Certain cases (improper name attack) cannot be solved this way
  - Also, false positives are possible
- Cai *et al.* showed
  - Guarantees require program knowledge [Oakland 2009]

Pathname Manipulators

- Users who can influence the result of a namespace resolution
  - Root users modify system namespace
  - Normal users modify their own namespace
- *U belongs to the manipulators of a name if the resolution of that name visits directories owned or writable by U*
- Be careful when others are manipulators
  - Programmers often make mistakes
  - So, implement a principled solution

Unsafe Subtrees

- Identify "unsafe subtrees" of the filesystem
- A directory is unsafe for a user if
  - anyone other than the user (or root) can write it
- Take precautions when using them
  - Resolve a pathname unit by unit
  - Enforce safe resolution conditions
- Directly focus on resolution

Safe and Unsafe Names

- A name is safe for some user if
  - only that user can manipulate it
- System safe:
  - Only manipulate by root
- Safe for U:
  - Only U and root can manipulate
- Unsafe
  - Otherwise

Options to Limit Risk

- Don’t open symbolic links
  - Prevents redirection to other subtrees
  - But, may need to use symbolic links
- Don’t open files with multiple hard links
  - Prevent good and bad guys from creating links
  - Easy denial of service
- Also, these defenses aren’t strong enough
  - What about resolutions in middle of pathname?

Safe-Open Property

- If a file has safe-names for U, then safe-open will not open it with unsafe names
- Assumes
  - Directory tree appears only once (no loop-back mounts)
  - Mounted in only safe locations (NFS)
  - Each directory has one parent
  - Good guys don't induce a race
- Proof: unsafe uses will be detected
  - Consider a file with safe and unsafe names, use unsafe
  - More than one hard link to file – arrive in unsafe mode
  - One hard link – either safe or would be blocked (no .. or symlink)

Implementation

- Extension to user-space library
  - Use *openat, readlinkat, fstatat* to perform reads using descriptors of directories rather than file names
  - Check each directory for “safety”
  - Prevent side effects
  - Include other safe operation, such as *safe-create*

Use

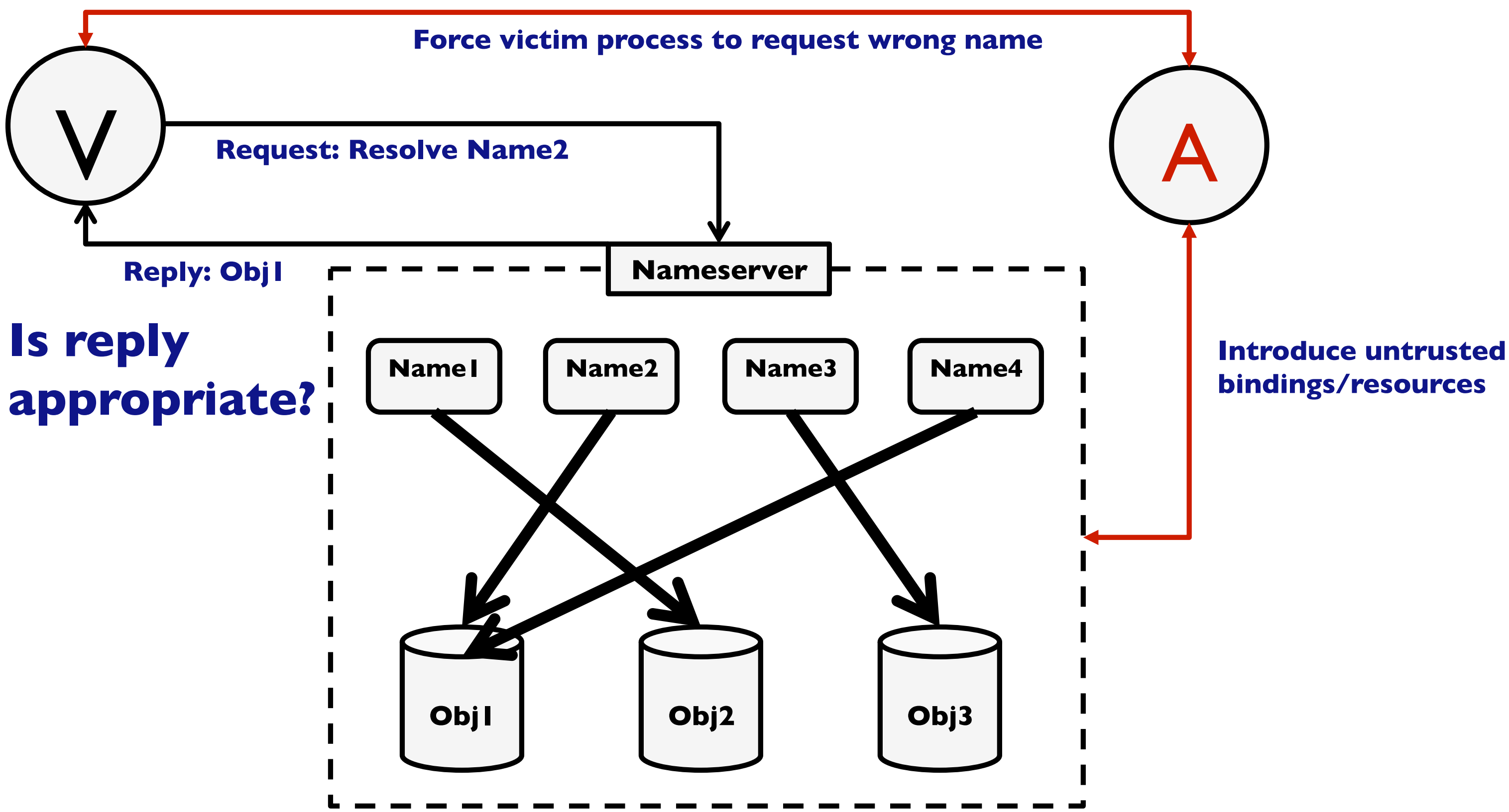
- Found vulnerabilities
  - CUPS – unprivileged process could replace file in shared directory
  - MySQL – creates a file as root in a directory owned by mysqld
  - HAL daemon – opens a file as root in a directory owned by hald
- Found policy issues (false positives)
  - Man pages – man user
  - Temporary directories – use ..
  - gdm – group write
- Web site
  - Lots of owners, so breaks by default (MAC has more principals)
  - Instead, restrict only if *file to be opened has another safe name*

System Defenses

- We have seen defenses against namespace resolution attacks
- Insight: All these enforce two invariants

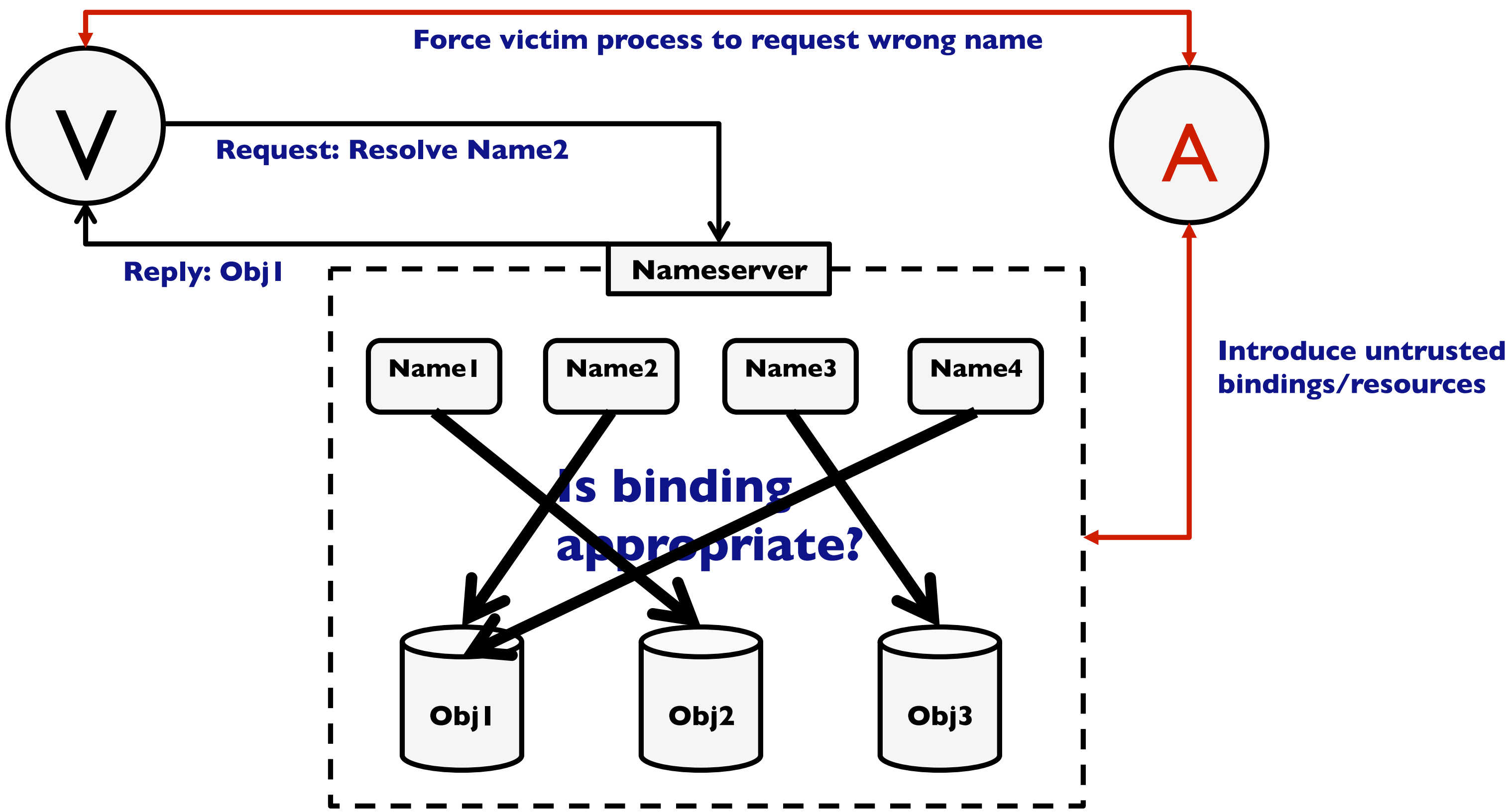
Invariant 1 - Resource

- i-resource(namespace, name, context)
  - Resource fetched for name in namespace is appropriate for that context



Invariant 2 - Binding

- i-binding(namespace, name, context)
  - Binding used to resolve name in namespace is appropriate for that context



Summary

PENNSTATE

- Namespace Resolution Attacks
  - Redirect the victim to another resource
- Lots of distinct attacks redirect victims
- Chari *et al.* describe a system-only defense using restrictions on the bindings accessed
  - Some limitations and false positives
- Cai *et al.* show that such limitations are inherent for redirection attacks
  - Some combination of false positives or missed attacks or program info needed

Questions

PENNSTATE 1855