Automatic Generation of Data-Oriented Exploits

Hong Hu, Zheng Leong Chua,
Sendroiu Adrian, Prateek Saxena, Zhenkai Liang

National University of Singapore

USENIX Security Symposium 2015, Washington, D.C., USA

Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray

Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP

Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP
- Defenses
  - Data Execution Prevention

Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP
- Defenses
  - Data Execution Prevention
  - Control Flow Integrity

Control Flow Attacks Are Getting Harder

- State-of-the-art exploits
  - Code injection
    - heap spray / JIT spray
  - Code reuse
    - ret2libc, ROP
    - control-flow bending
- Defenses
  - Data Execution Prevention
  - Control Flow Integrity

- Stat-of-the-art exploits from memory errors
  - Code injection (e.g., heap spray / JIT spray)
  - Code reuse (e.g., ret2libc, ROP)
- Defenses
  - DEP, CFI, ASLR
  - Block control flow hijacking in principle

CONTROL PLANE


DATA PLANE
Defenses
DEP, CFI, ASLR
Block control flow hijacking in principle
CONTROL PLANE

Data-Oriented Exploits

- State-of-the-art: Corrupt security-critical data
  - leave control flow as the same
  - Exhibit “significant” damage

Data-Oriented Exploits

- State-of-the-art: Corrupt security-critical data
  - leave control flow as the same
  - Exhibit “significant” damage

```
// set root privilege
seteuid(0);
......
// set normal user privilege
seteuid(pw->pw_uid);
// execute user’s command
```

Wu-ftpd *setuid* operation*

* Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-Control-Data Attacks Are Realistic Threats. In USENIX 2005.

Data-Oriented Exploits

- State-of-the-art: Corrupt security-critical data
  - leave control flow as the same
  - Exhibit "significant" damage

```
// set root privilege
seteuid(0);
......
// set normal user privilege
seteuid(pw->pw_uid);
// execute user's command
```

Wu-ftpd *setuid* operation*

```
//0x1D4, 0x1E4 or 0x1F4 in JScript 9,
//0x188 or 0x184 in JScript 5.8,
safemode = *(DWORD *)(jsobj + 0x188);
if( safemode & 0xB == 0 ) {
        Turn_on_God_Mode();
}
```

IE *SafeMode* Bypass+

* Shuo Chen, Jun Xu, Emre C. Sezer, Prachi Gauriar, and Ravishankar K. Iyer. Non-Control-Data Attacks Are Realistic Threats. In USENIX 2005.

\+ Yang Yu. Write Once, Pwn Anywhere. In Black Hat USA 2014

Data-Oriented Exploits

- State-of-the-art: Corrupt security-critical data
  - leave control flow as the same
  - Exhibit “significant” damage

Contributions

Contributions

- New class of Data-Oriented Exploits
  - Reuses existing *data flows* in normal execution
  - Agnostic to CFI, DEP and often ASLR

Contributions

- New class of Data-Oriented Exploits
  - Reuses existing *data flows* in normal execution
  - Agnostic to CFI, DEP and often ASLR

Contributions

- New class of Data-Oriented Exploits
  - Reuses existing *data flows* in normal execution
  - Agnostic to CFI, DEP and often ASLR

Contributions

- New class of Data-Oriented Exploits
  - Reuses existing *data flows* in normal execution
  - Agnostic to CFI, DEP and often ASLR
- Data Flow Stitching
  - Systematic search for data-oriented exploits
  - Works on binary directly

Contributions

- New class of Data-Oriented Exploits
  - Reuses existing *data flows* in normal execution
  - Agnostic to CFI, DEP and often ASLR
- Data Flow Stitching
  - Systematic search for data-oriented exploits
  - Works on binary directly
- Results
  - Concrete exploits on real web/file servers
  - 19 exploits (16 new) from 8 vulnerabilities

Motivating Example

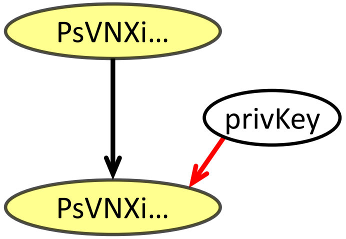
- SSL-enabled web server

```
int server() {
 char *userInput, *fileName;
 char *privKey, *result, output[BUFSIZE];
 char fullPath[BUFSIZE]="/path/to/root/";

 privKey=loadPrivKey("/path/to/privKey");
 GetConnection(privKey, ...);
 userInput = read_socket();
 if (checkInput(userInput)) {
  fileName = getFileName(userInput);
  strcat(fullPath,  fileName);
  result = retrieve(fullPath);
  sprintf(output, “%s:%s”, fileName, result);
  sendOut(output);
 }
}
```

Motivating Example

- SSL-enabled web server

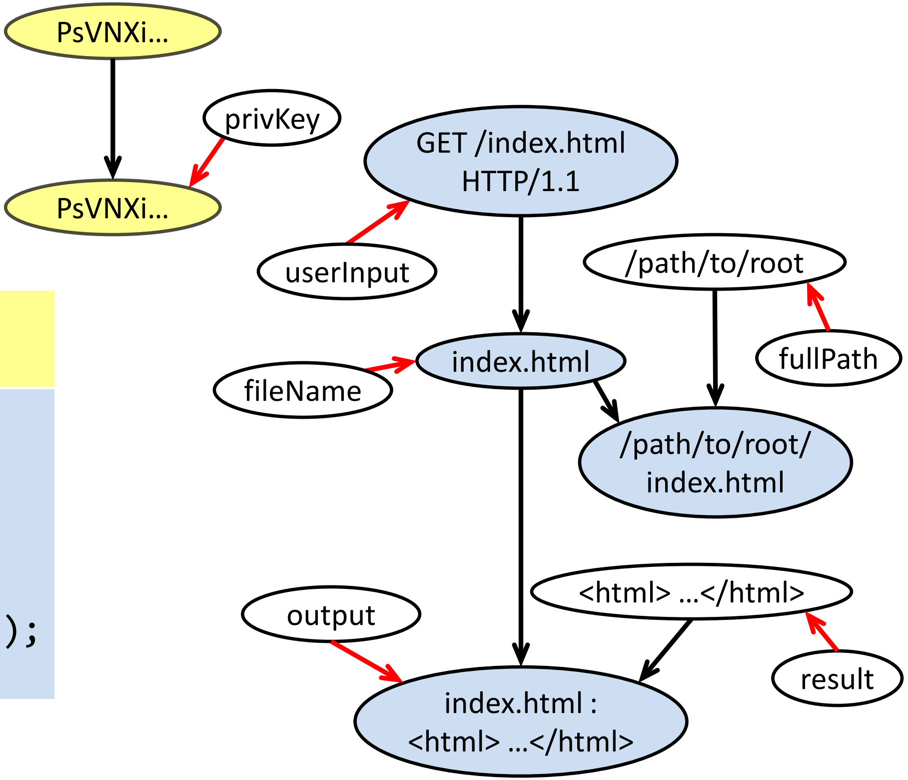
```
int server() {
 char *userInput, *fileName;
 char *privKey, *result, output[BUFSIZE];
 char fullPath[BUFSIZE]="/path/to/root/";

 privKey=loadPrivKey("/path/to/privKey");
 GetConnection(privKey, ...);
 userInput = read_socket();
 if (checkInput(userInput)) {
  fileName = getFileName(userInput);
  strcat(fullPath,  fileName);
  result = retrieve(fullPath);
  sprintf(output, “%s:%s”, fileName, result);
  sendOut(output);
 }
}
```



Motivating Example

- SSL-enabled web server

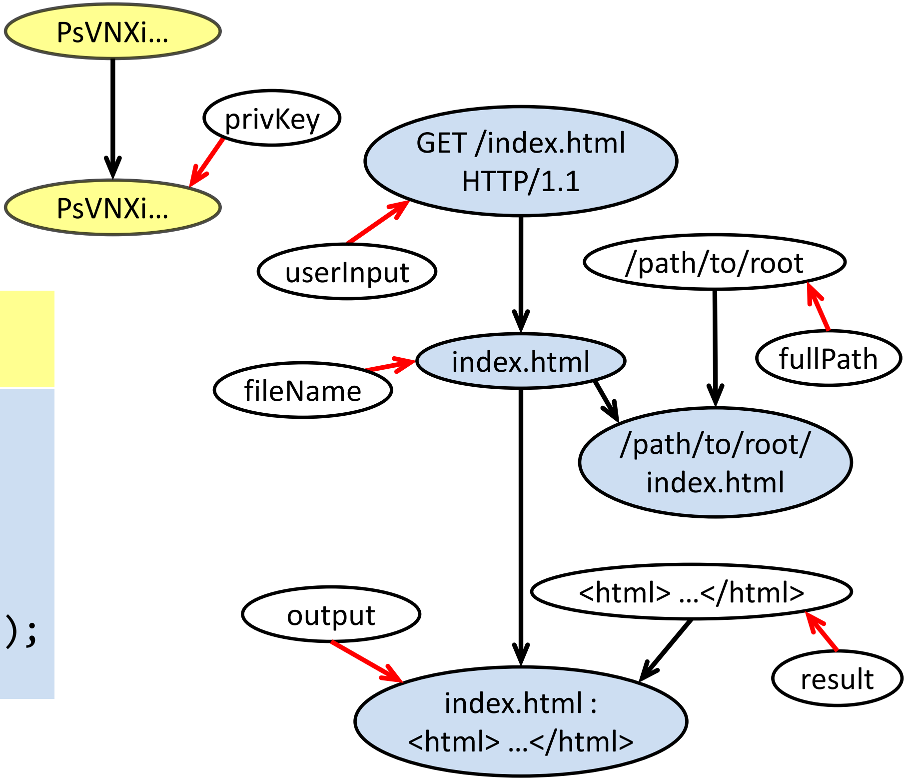
```c
int server() {
 char *userInput, *fileName;
 char *privKey, *result, output[BUFSIZE];
 char fullPath[BUFSIZE]="/path/to/root/";

 privKey=loadPrivKey("/path/to/privKey");
 GetConnection(privKey, ...);
 userInput = read_socket();
 if (checkInput(userInput)) {
  fileName = getFileName(userInput);
  strcat(fullPath,  fileName);
  result = retrieve(fullPath);
  sprintf(output, "%s:%s", fileName, result);
  sendOut(output);
 }
}
```



Motivating Example

- SSL-enabled web server

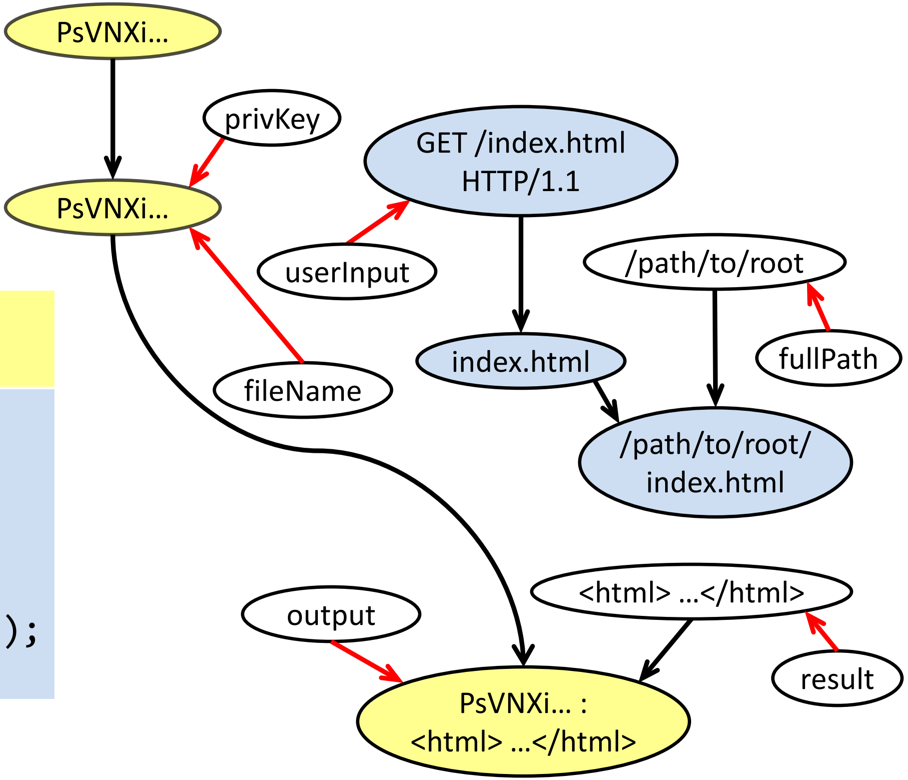
```
int server() {
 char *userInput, *fileName;
 char *privKey, *result, output[BUFSIZE];
 char fullPath[BUFSIZE]="/path/to/root/";

 privKey=loadPrivKey("/path/to/privKey");
 GetConnection(privKey, ...);
 userInput = read_socket();
 if (checkInput(userInput)) {
  fileName = getFileName(userInput);
  strcat(fullPath,  fileName);
  result = retrieve(fullPath);
  sprintf(output, “%s:%s”, fileName, result);
  sendOut(output);
 }
}
```



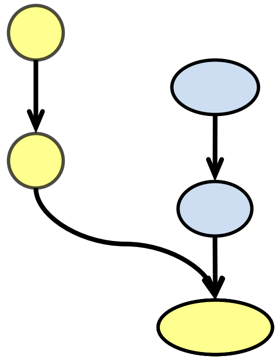
Motivating Example

- SSL-enabled web server

```
int server() {
 char *userInput, *fileName;
 char *privKey, *result, output[BUFSIZE];
 char fullPath[BUFSIZE]="/path/to/root/";

  privKey=loadPrivKey("/path/to/privKey");
  GetConnection(privKey, ...);
  userInput = read_socket();
  if (checkInput(userInput)) {
   fileName = getFileName(userInput);
   strcat(fullPath,  fileName);
   result = retrieve(fullPath);
   sprintf(output, “%s:%s”, fileName, result);
   sendOut(output);
  }
}
```



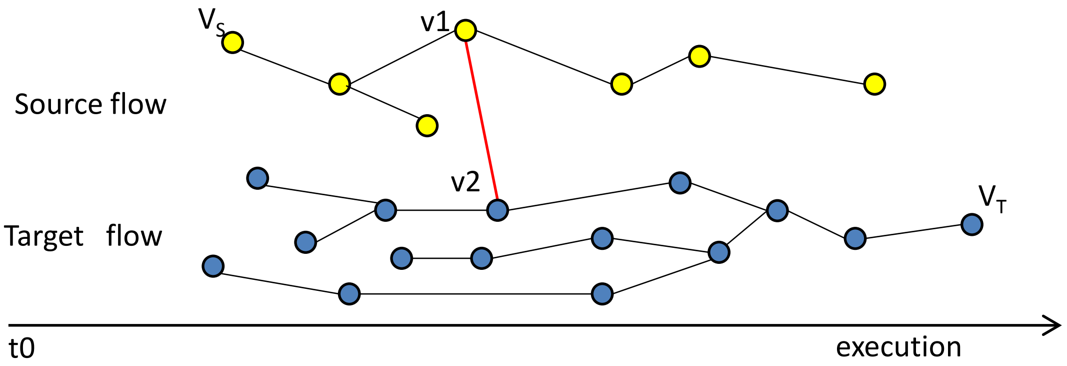
Data-Flow Stitching



- Manipulate data flows for exploits
- Enables systematic way to search for exploits
  - Input: binary & error-exhibiting input
  - Output: data-oriented exploits
- Goal:
  - Information Leakage (e.g., password, keys)
  - Privilege Escalation (e.g., setuid, access priv. files)
- Constraints:
  - Keep the control-flow same
  - Prevent abrupt termination
  - No knowledge of randomized values (CFI tags, ASLR addresses)

Challenges

- Time-consuming search
  - The search-space: Cartesian product |SrcFlow| ***X*** |TgtFlow|
  - Heavy analysis for each candidate



- Our solution:
  - Filter out candidates with memory error influence
  - Use an SMT solver to verify candidates

Single-Edge Stitch

- Corrupt data vertex

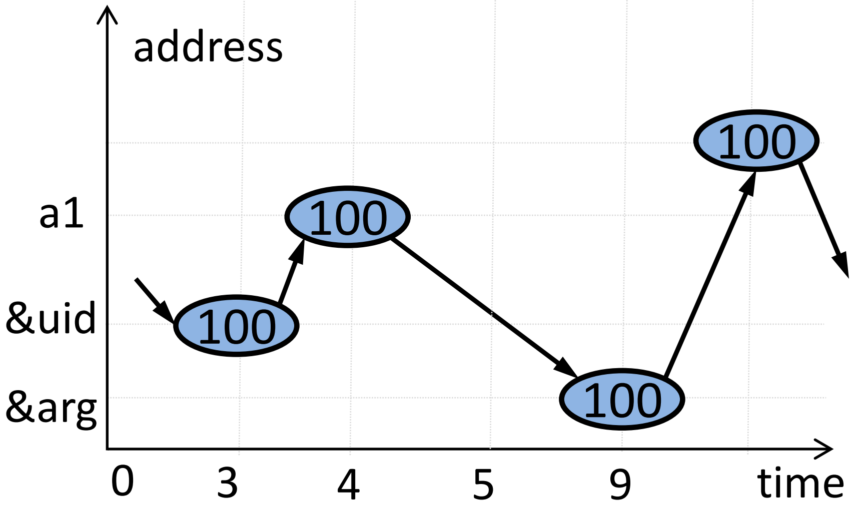
```
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);  //set root uid
...
seteuid(pw->pw_uid); //set normal uid
...
```

Single-Edge Stitch

- Corrupt data vertex

```
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);   //set root uid
...
seteuid(pw->pw_uid); //set normal uid
...
```

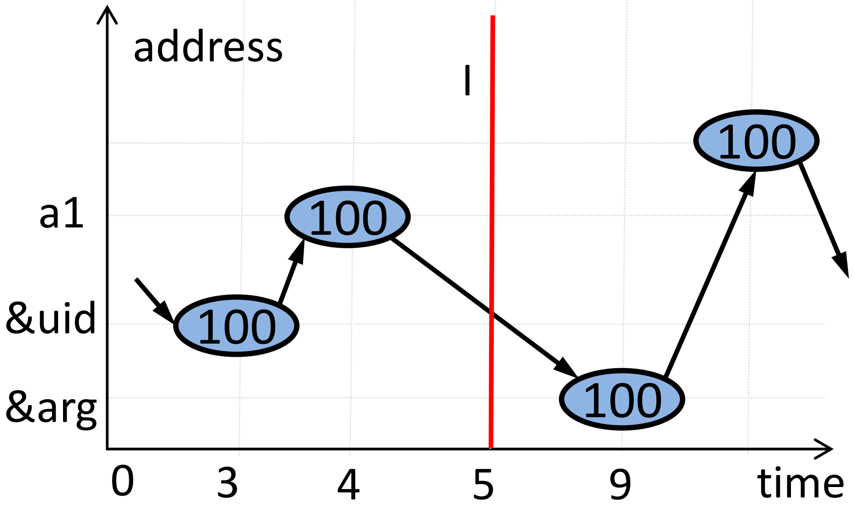
- 2D-DFG



Single-Edge Stitch

- Corrupt data vertex

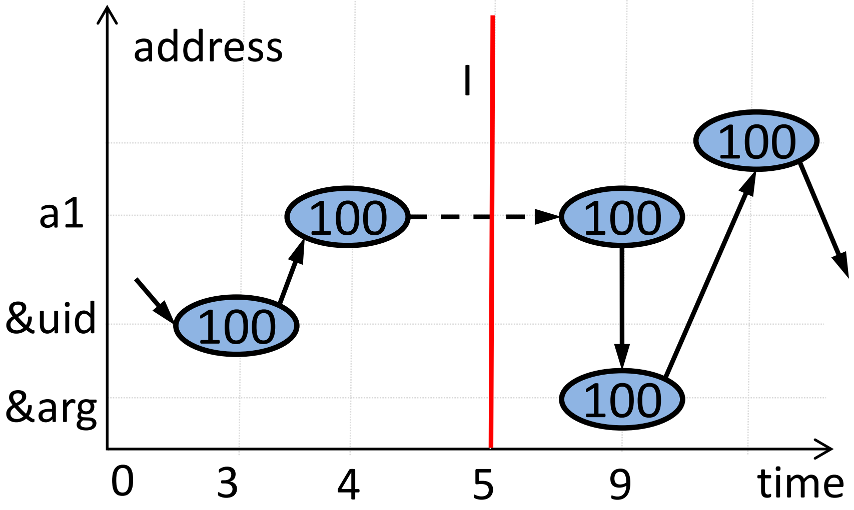
```
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);   //set root uid
...
seteuid(pw->pw_uid); //set normal uid
...
```



Single-Edge Stitch

- Corrupt data vertex

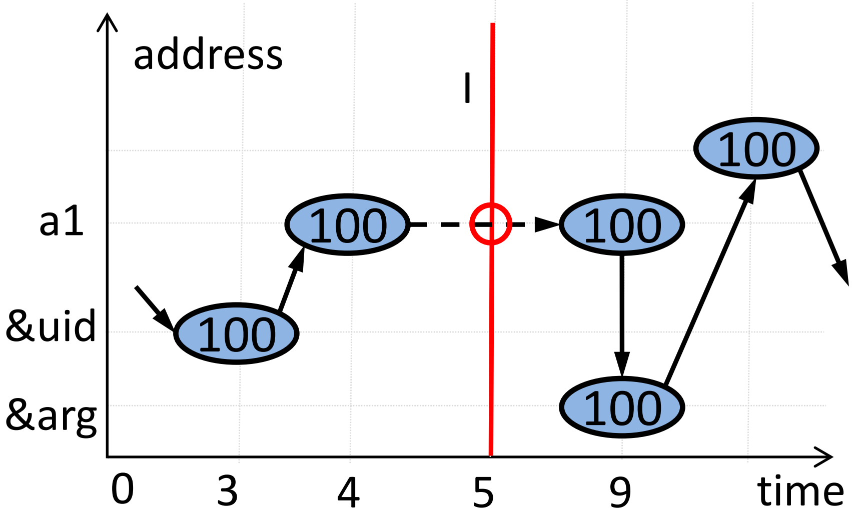
```
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);  //set root uid
...
seteuid(pw->pw_uid); //set normal uid
...
```



Single-Edge Stitch

- Corrupt data vertex

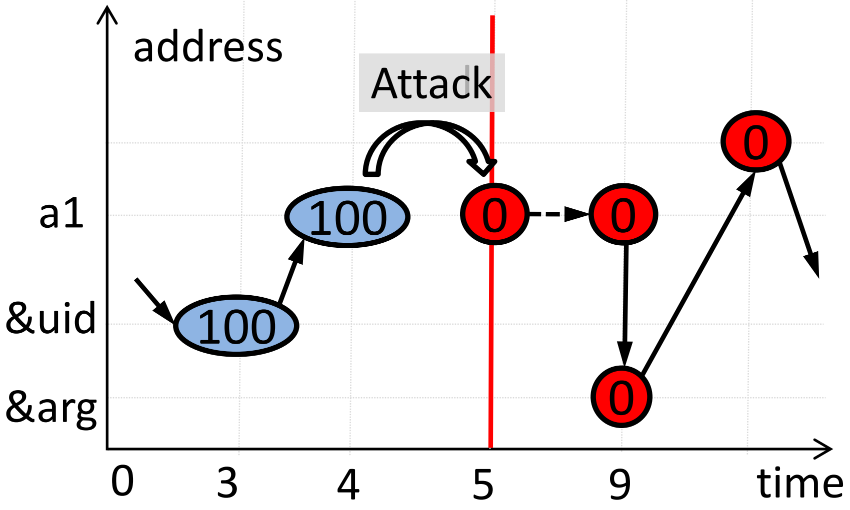
```
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);  //set root uid
...
seteuid(pw->pw_uid); //set normal uid
...
```



Single-Edge Stitch

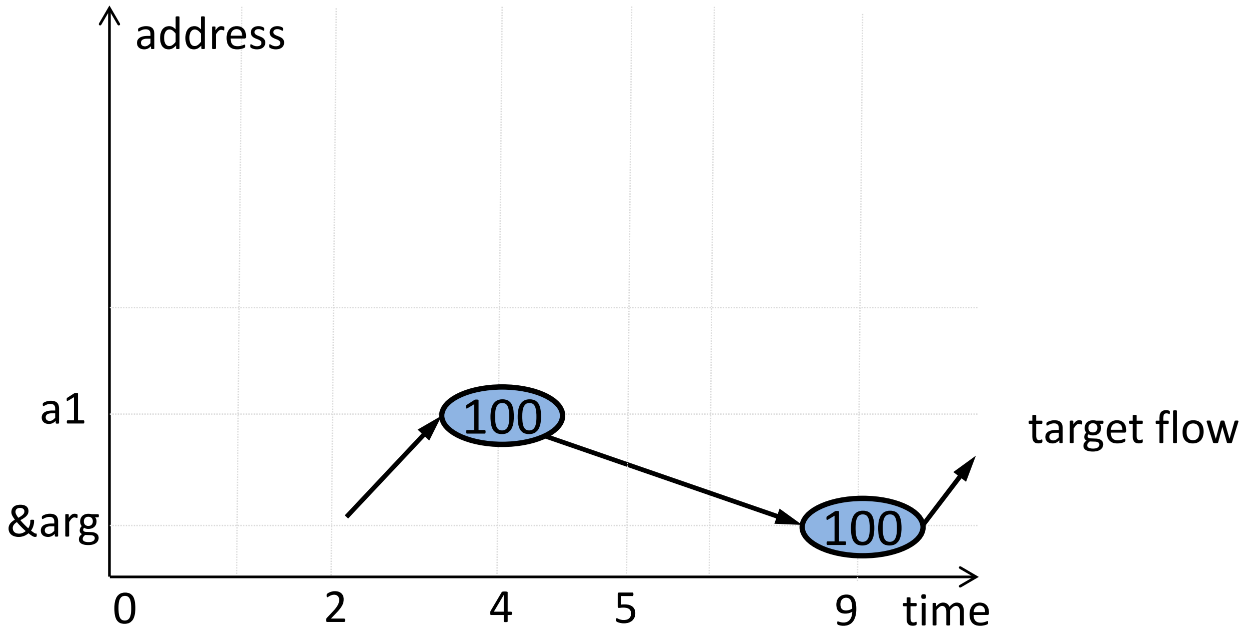
- Corrupt data vertex

```
struct passwd {uid_t pw_uid; ... } pw;
...
int uid = getuid();
pw->pw_uid = uid;
printf(...); //format string error
...
seteuid(0);  //set root uid
...
seteuid(pw->pw_uid); //set normal uid
...
```



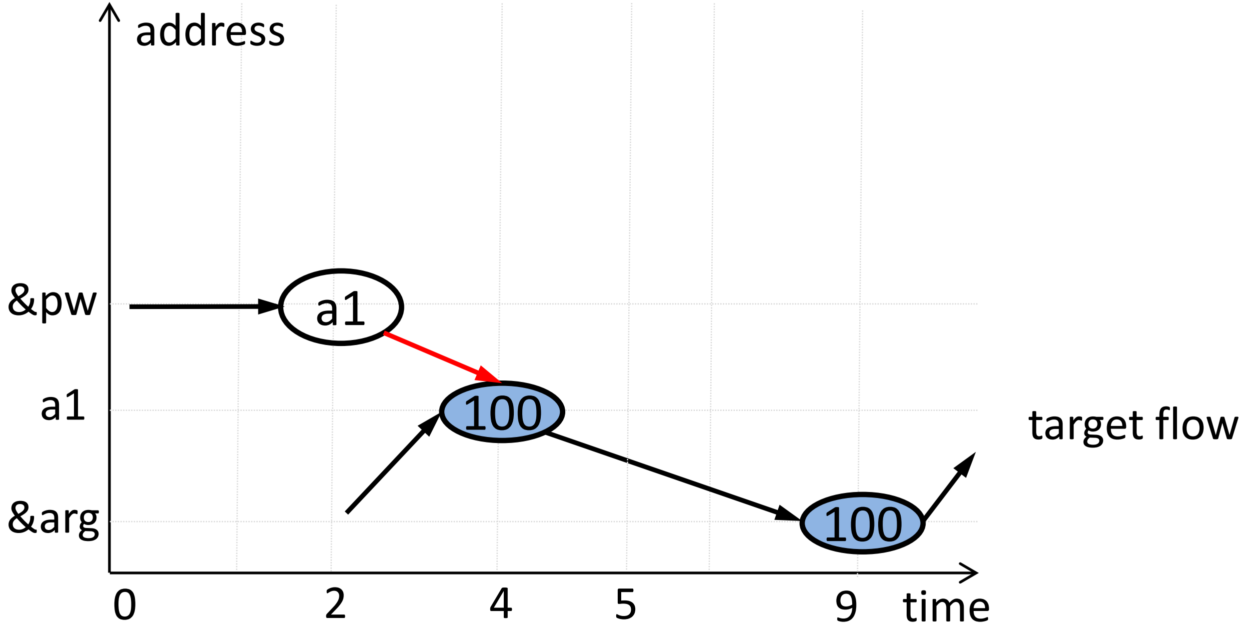
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



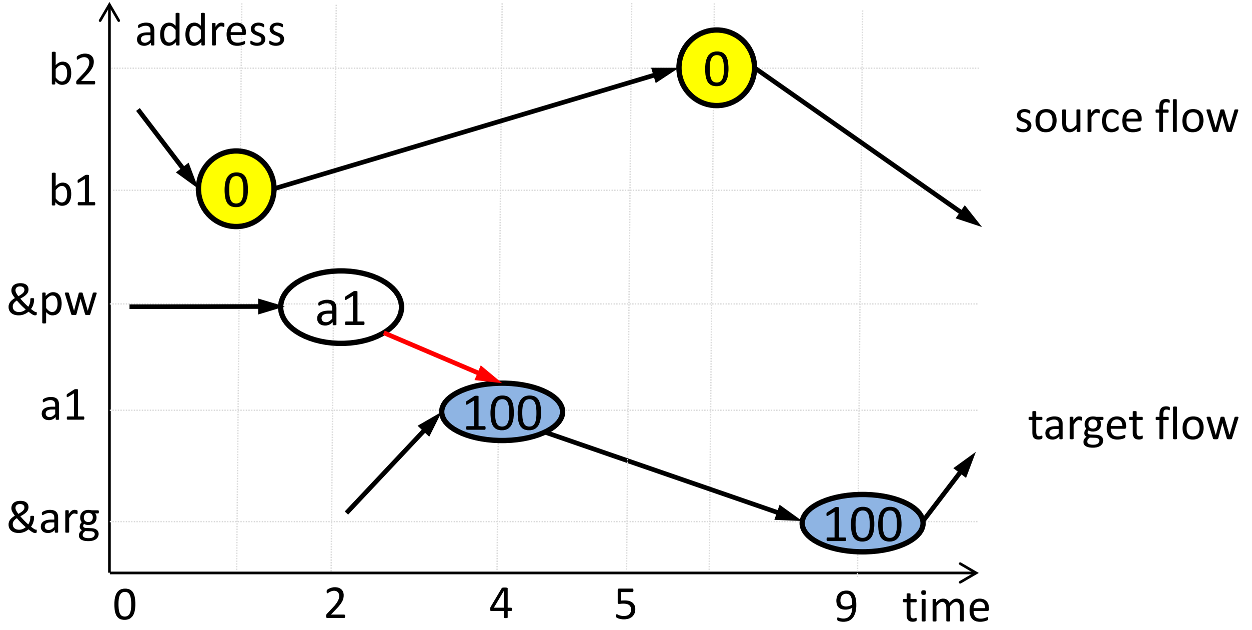
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



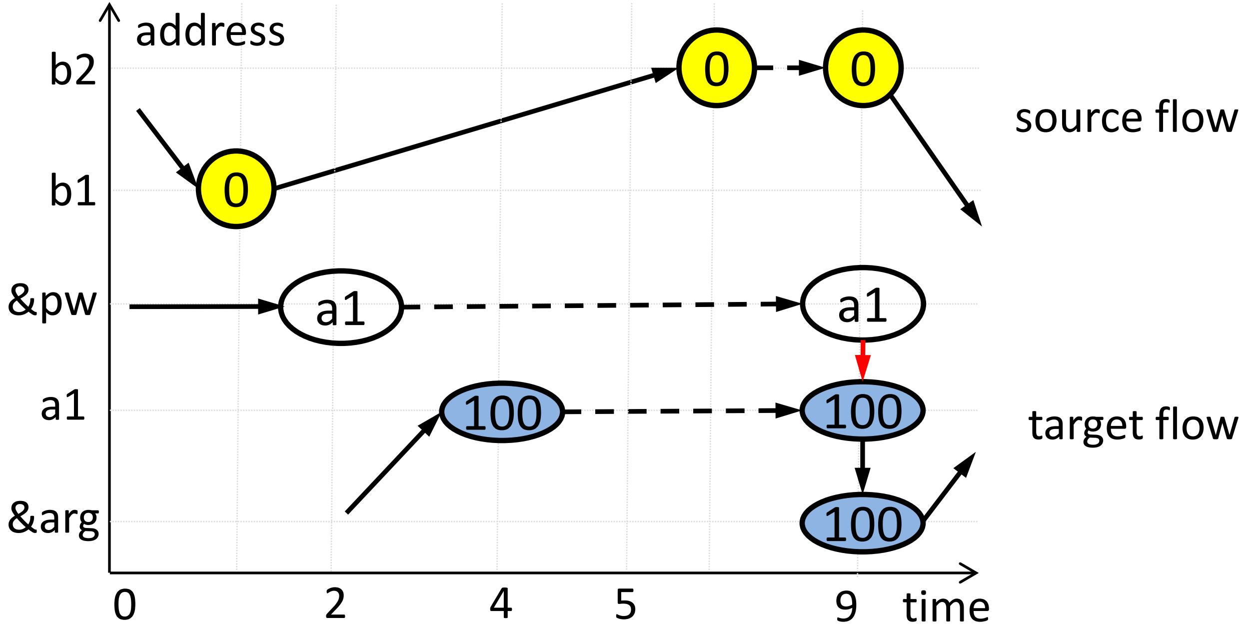
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



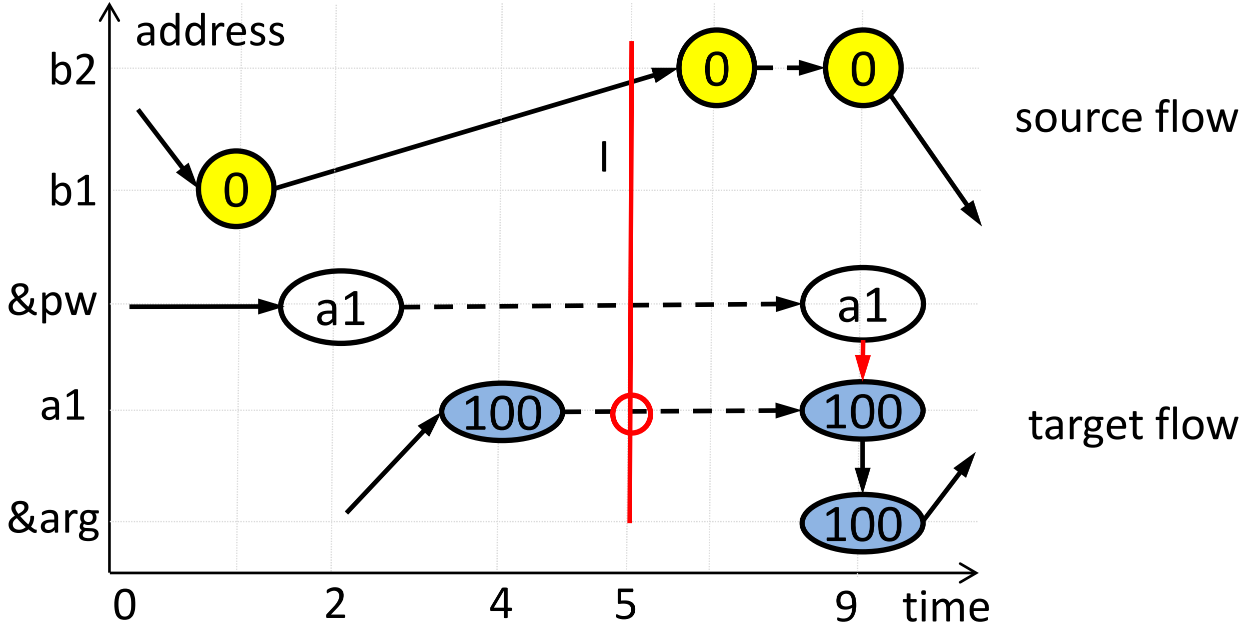
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



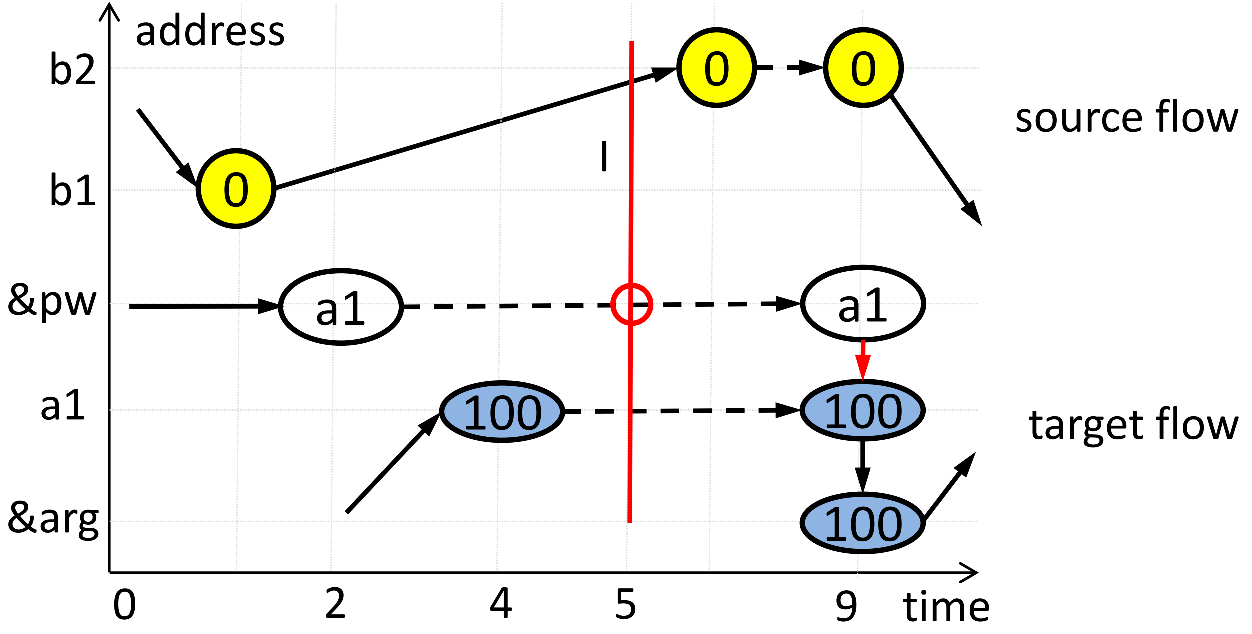
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



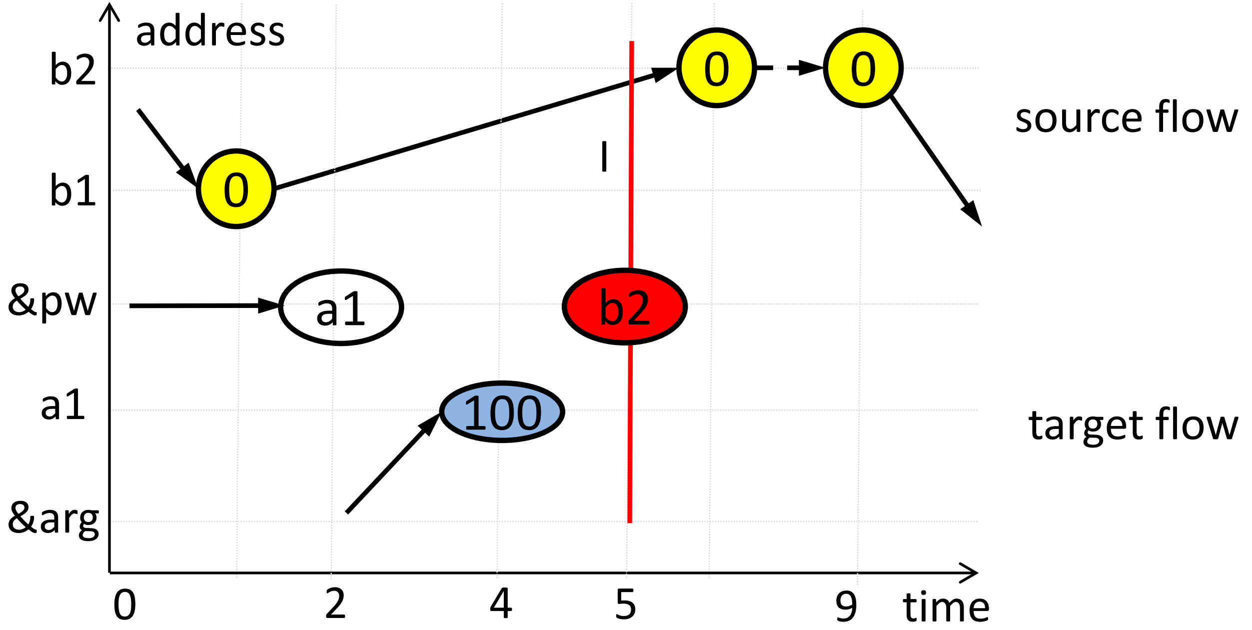
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



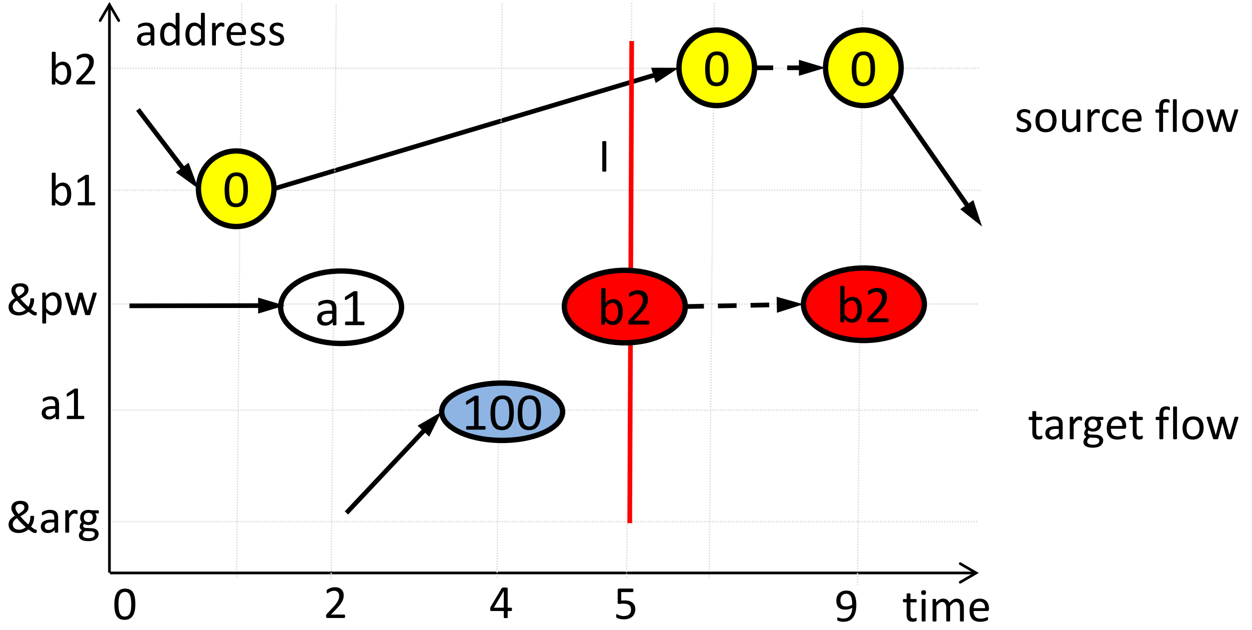
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



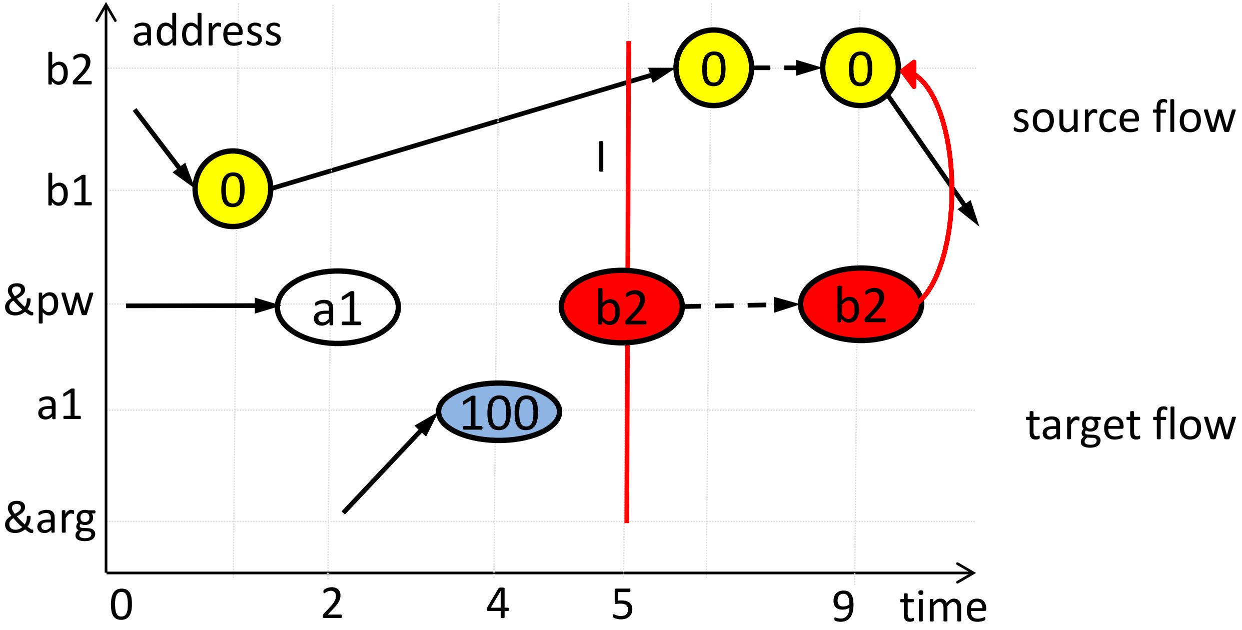
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



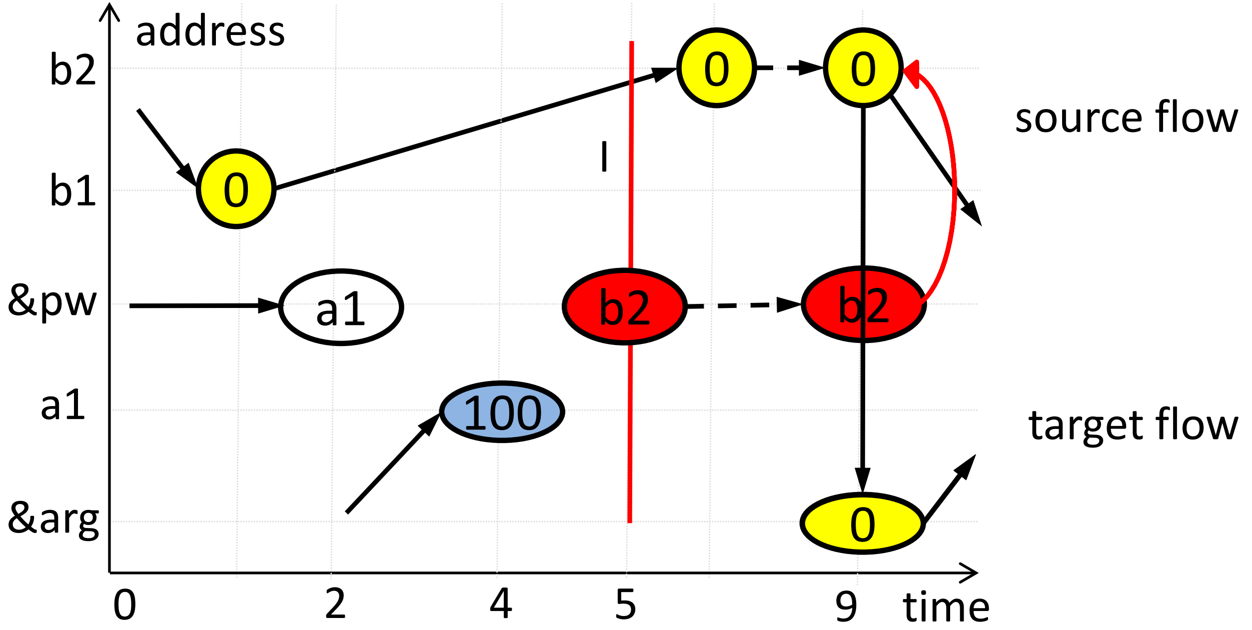
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



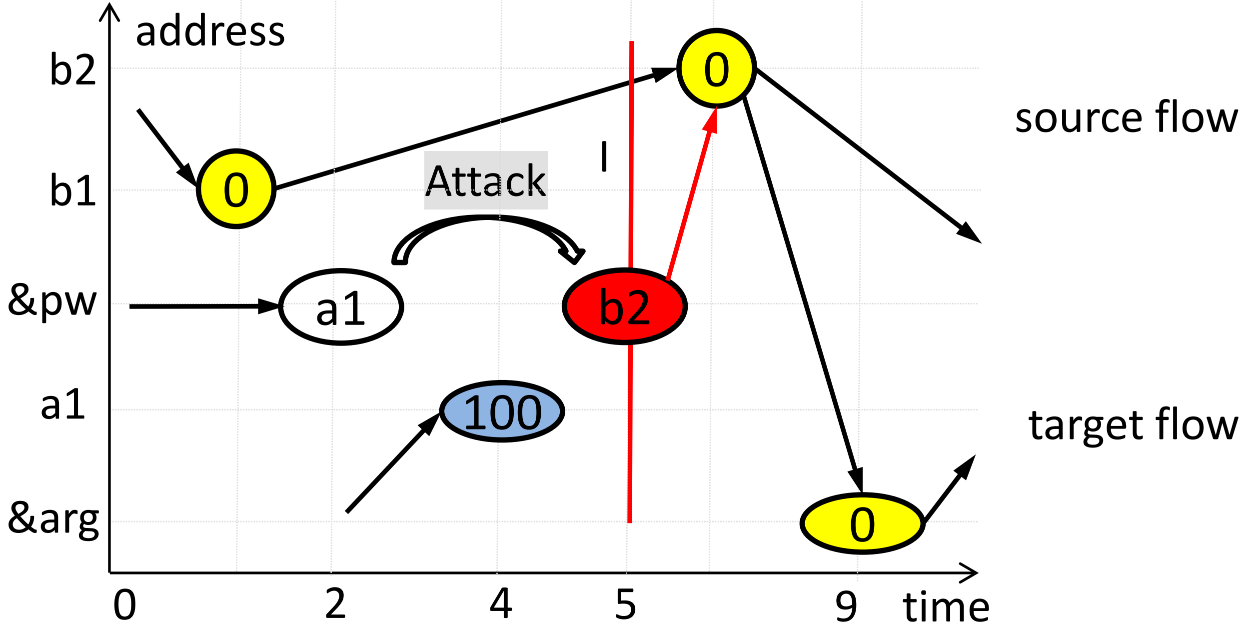
Pointer Stitch

- Corrupt pointers to connect data flows
  - Pointers decide data movement direction



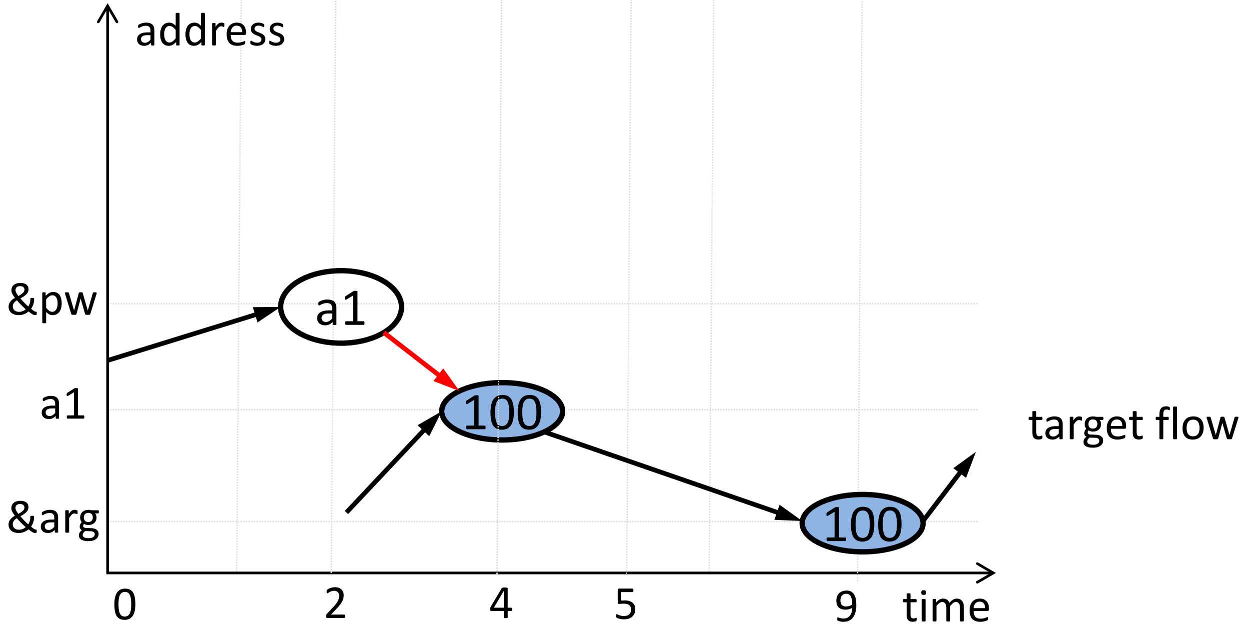
Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



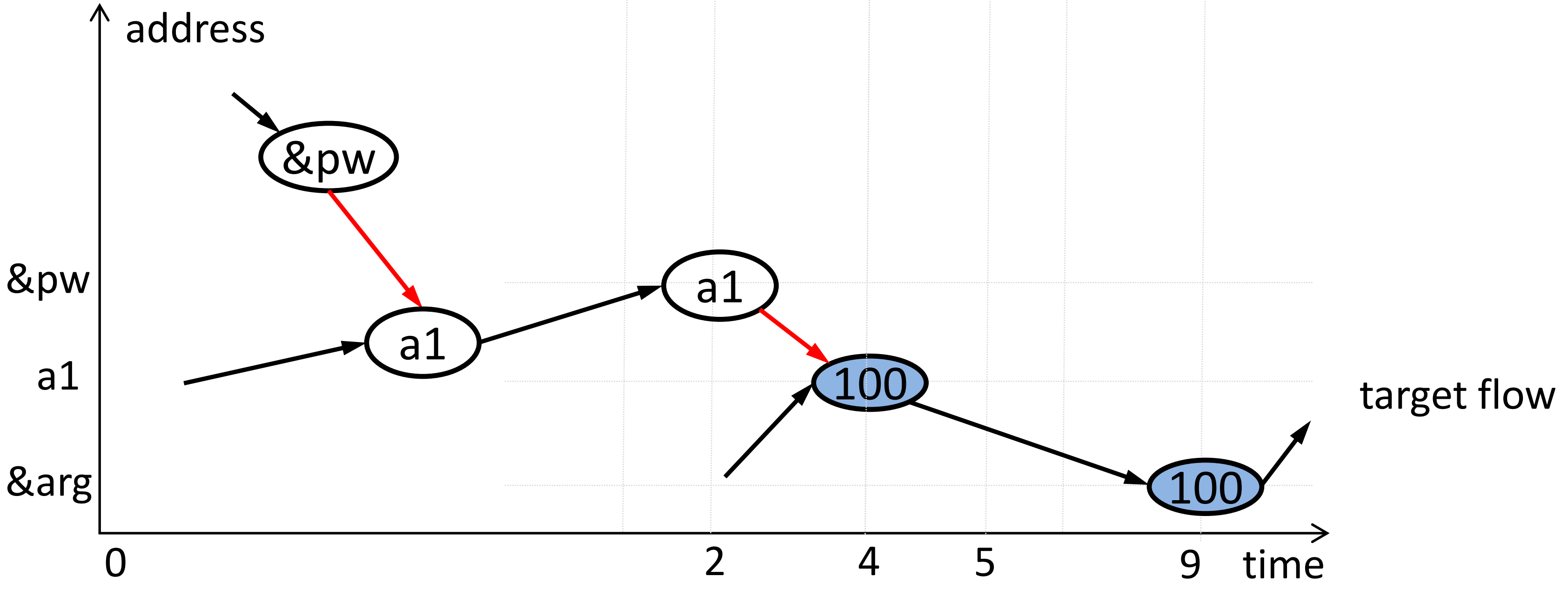
Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



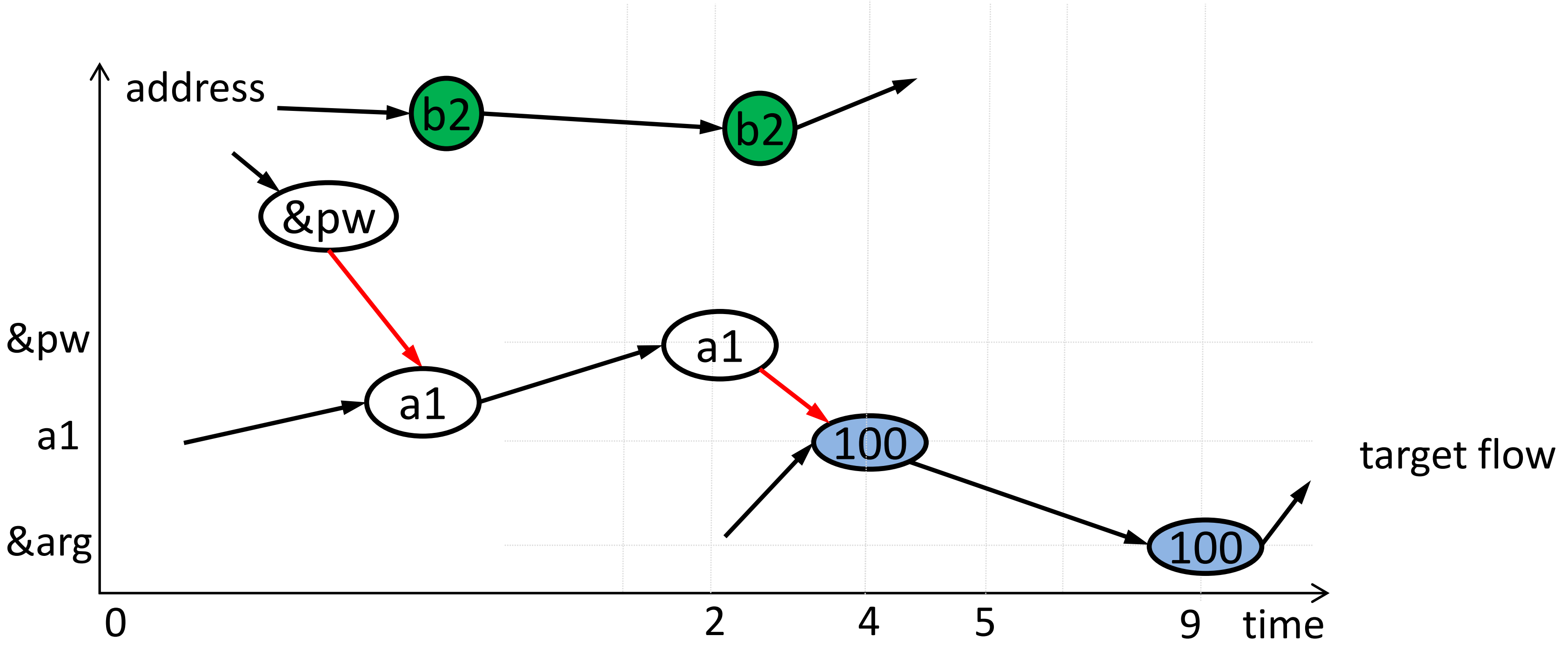
Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



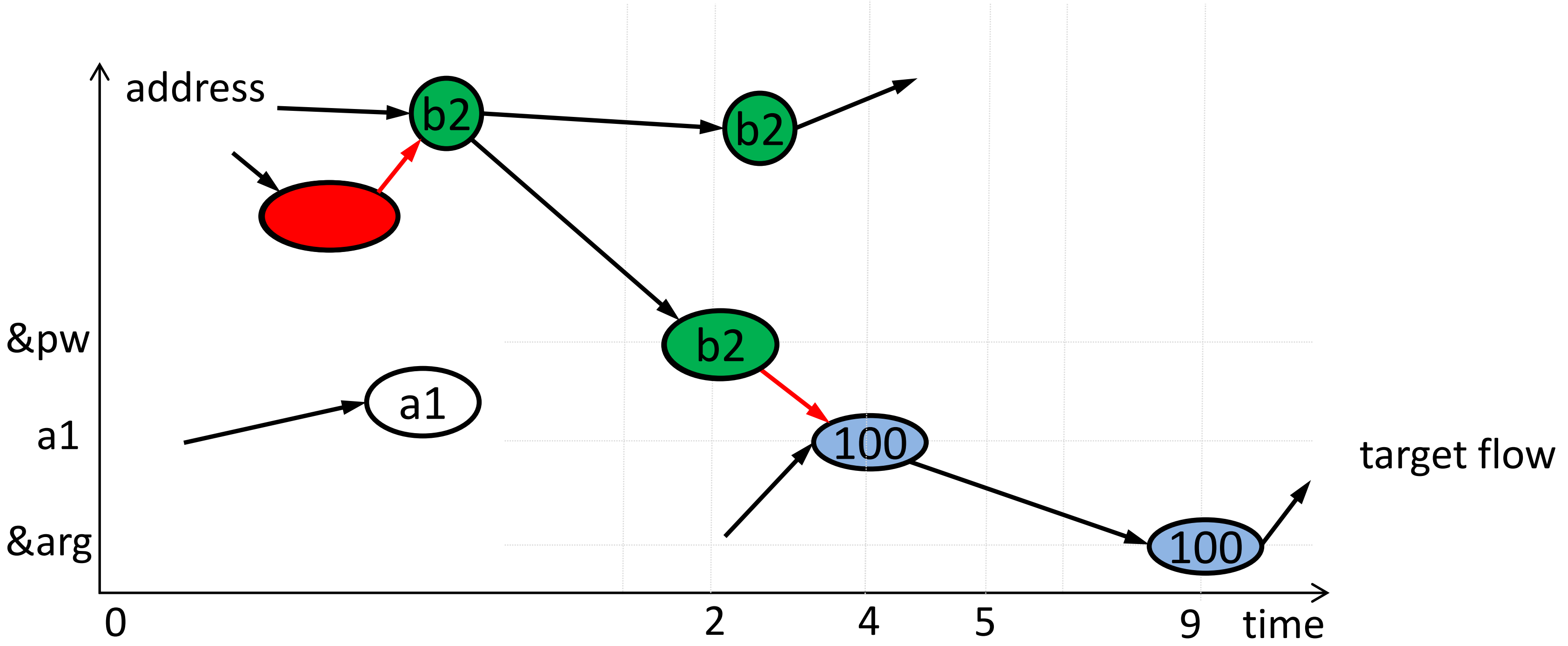
Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



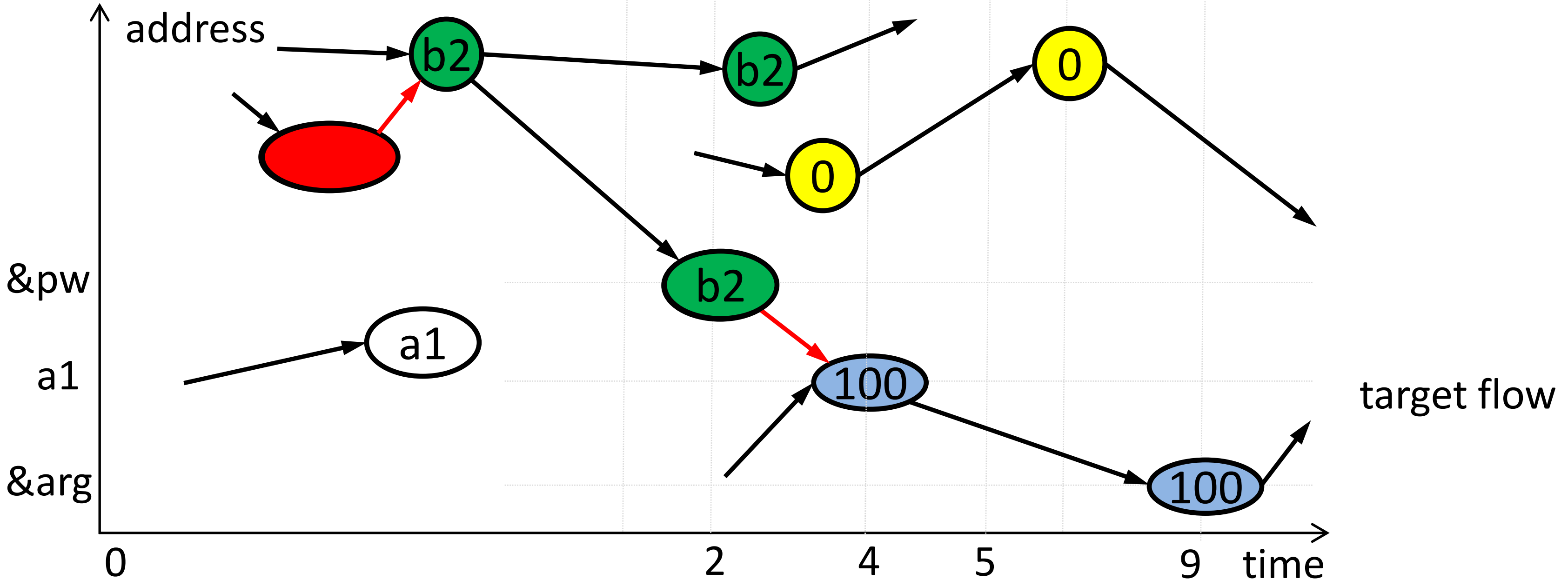
Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



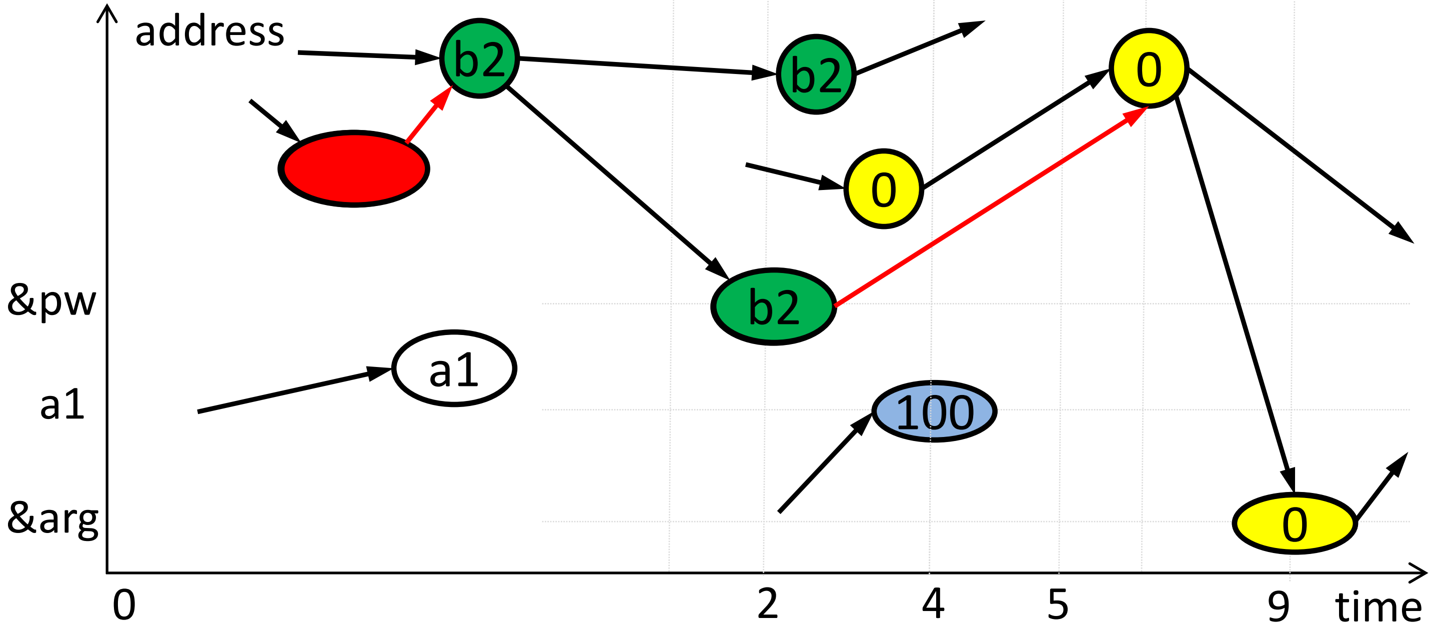
Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



Pointer Stitch

- Pointer Stitch corrupts pointer *vp*
  - *(*vp*) ---> target / source vertex



More Ways of Stitches

- 2-level stitch corrupts pointer vp_2
  - $*(*(vp_2))$ ---> $*(vp)$ ---> target / source vertex

More Ways of Stitches

- 2-level stitch corrupts pointer vp_2
  - $*(*(vp_2))$ ---> $*(vp)$ ---> target / source vertex
- N-level stitch corrupts pointer vp_N
  - $*(*(\ldots(vp_N)\ldots))$ ---> target / source vertex
  - Recursively invoke pointer stitch N times
  - Stitch Alignment
    - vp_N ---> vp_N' so that $*(*(\ldots(vp_N')\ldots))$ is the source / target vertex

More Ways of Stitches

- 2-level stitch corrupts pointer vp_2
  - $*(*(vp_2))$ ---> $*(vp)$ ---> target / source vertex
- N-level stitch corrupts pointer vp_N
  - $*(*(\ldots(vp_N)\ldots))$ ---> target / source vertex
  - Recursively invoke pointer stitch N times
  - Stitch Alignment
    - vp_N ---> vp_N' so that $*(*(\ldots(vp_N')\ldots))$ is the source / target vertex
- Multi-flow stitching
  - Intermediate data flows
  - Source flow -> flow 1 -> flow 2 -> … -> Target flow

Defeat ASLR --- Address Reuse

- Partial resue: offset is fixed

```
//attackers control %eax
mov (%esi,%eax,4), %ebx
mov %ecx, (%edi,%eax,4)
```

Defeat ASLR --- Address Reuse

- Partial resue: offset is fixed

```
//attackers control %eax
mov (%esi,%eax,4), %ebx
mov %ecx, (%edi,%eax,4)
```

- Complete reuse:
  – randomized address in memory

```
//attacker controls %eax
mov (%esi, %eax, 4), %ebx
mov %ecx, (%ebx)
mov (%ebx), %ecx
```

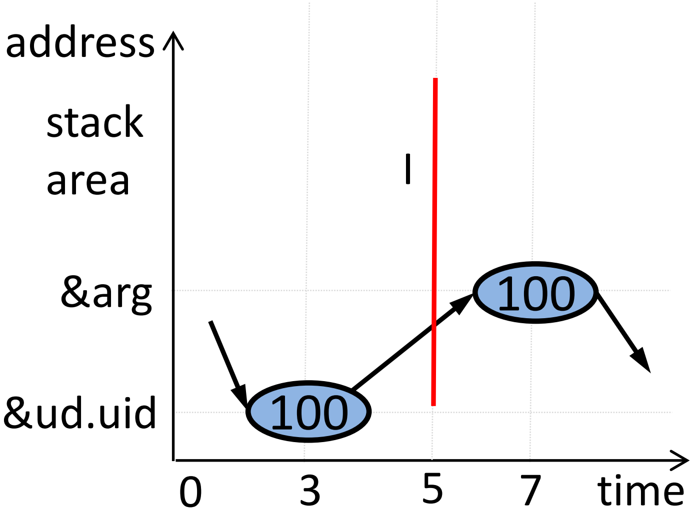
Defeat ASLR --- Address Reuse

- Partial resue: offset is fixed

//attackers control %eax
mov (%esi,%eax,4), %ebx
mov %ecx, (%edi,%eax,4)

- Complete reuse:
  - randomized address in memory

//attacker controls %eax mov (%esi, %eax, 4), %ebx
mov %ecx, (%ebx)
mov (%ebx), %ecx



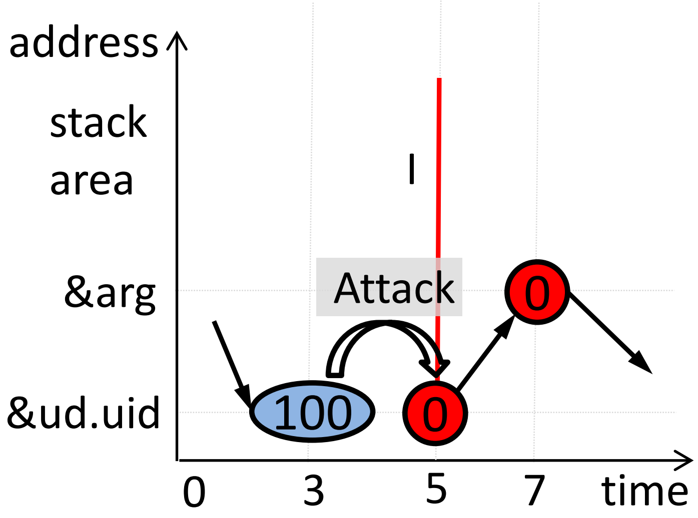
Defeat ASLR --- Address Reuse

- Partial resue: offset is fixed

```
//attackers control %eax
mov (%esi,%eax,4), %ebx
mov %ecx, (%edi,%eax,4)
```

- Complete reuse:
  - randomized address in memory

```
//attacker controls %eax
mov (%esi, %eax, 4), %ebx
mov %ecx, (%ebx)
mov (%ebx), %ecx
```



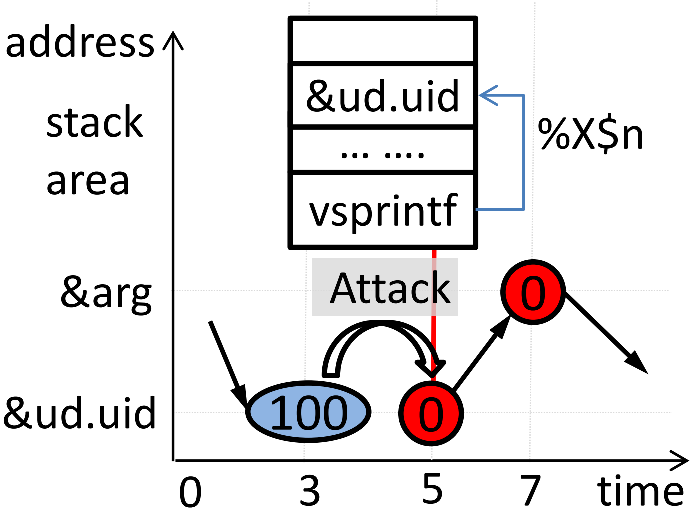
Defeat ASLR --- Address Reuse

- Partial resue: offset is fixed

```
//attackers control %eax
mov (%esi,%eax,4), %ebx
mov %ecx, (%edi,%eax,4)
```

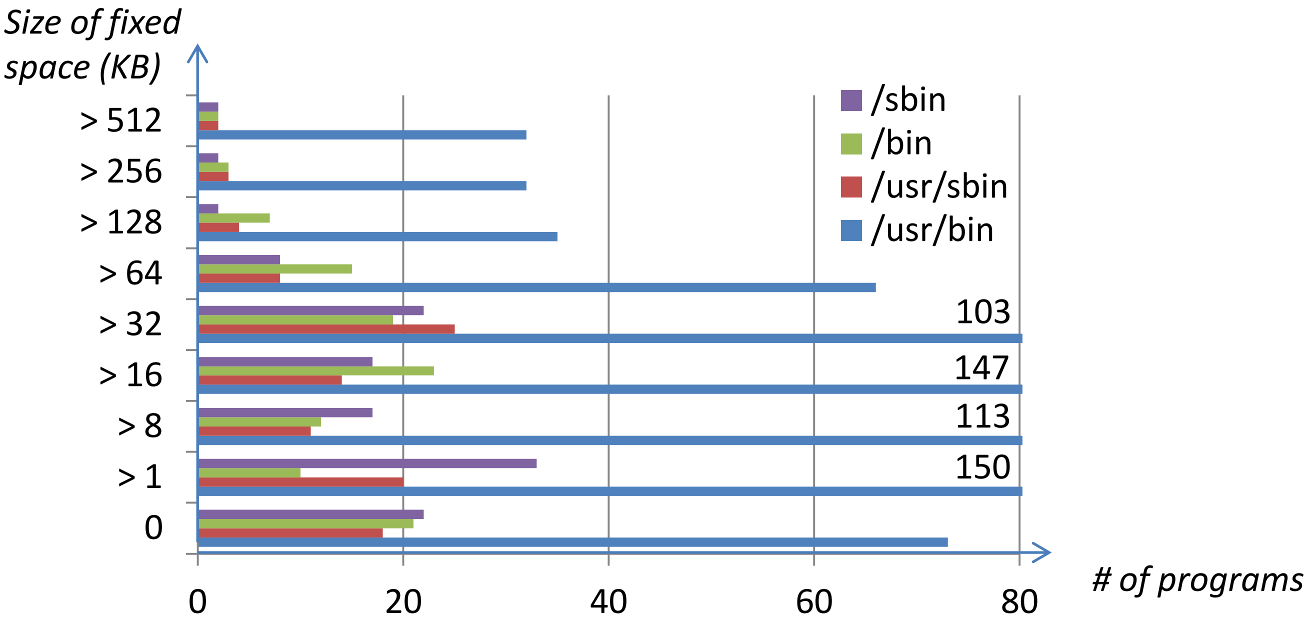
- Complete reuse:
  - randomized address in memory

```
//attacker controls %eax
mov (%esi, %eax, 4), %ebx
mov %ecx, (%ebx)
mov (%ebx), %ecx
```



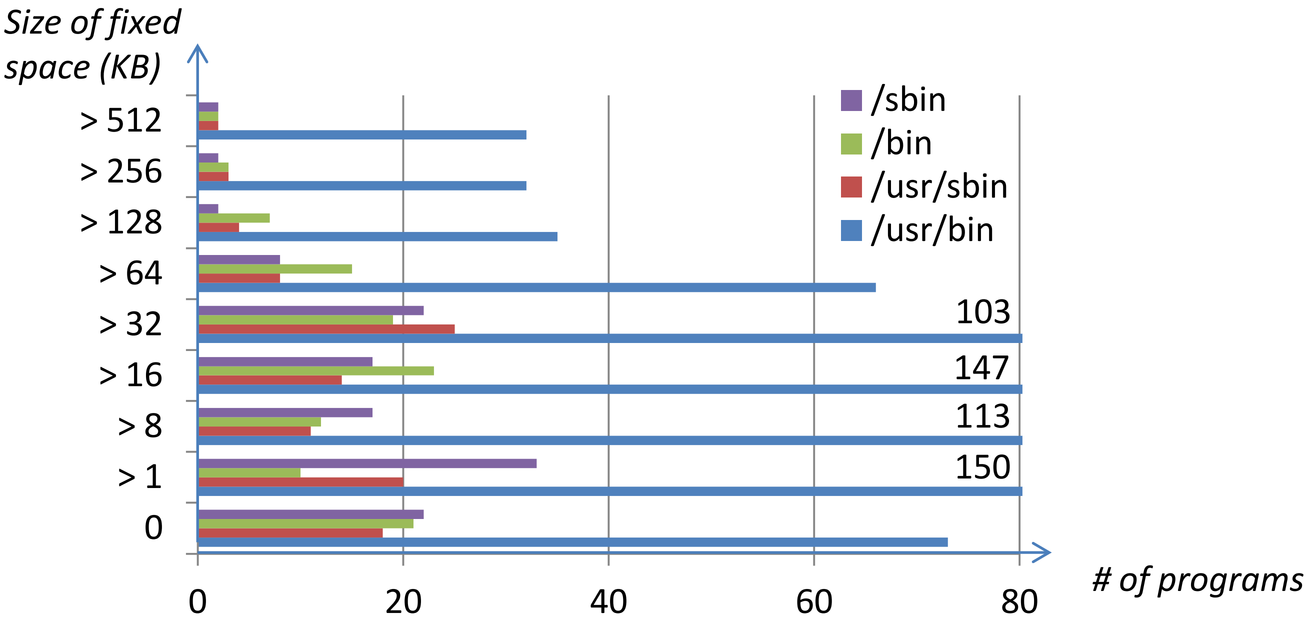
Stitch with ASLR

- Target deterministic addresses
  - non-PIE binaries on Linux



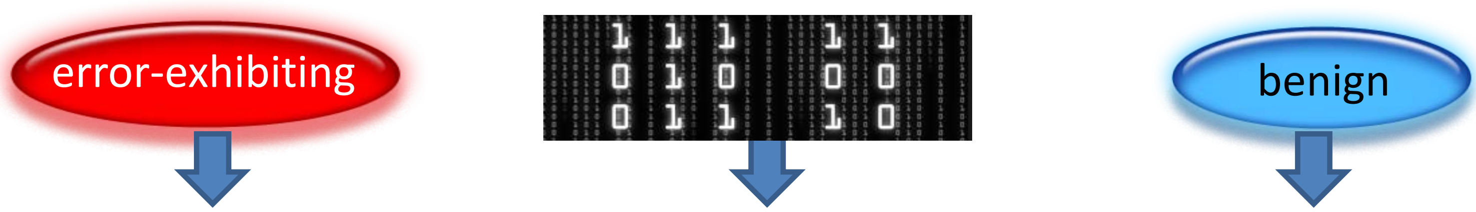
Stitch with ASLR

- Target deterministic addresses
  - non-PIE binaries on Linux

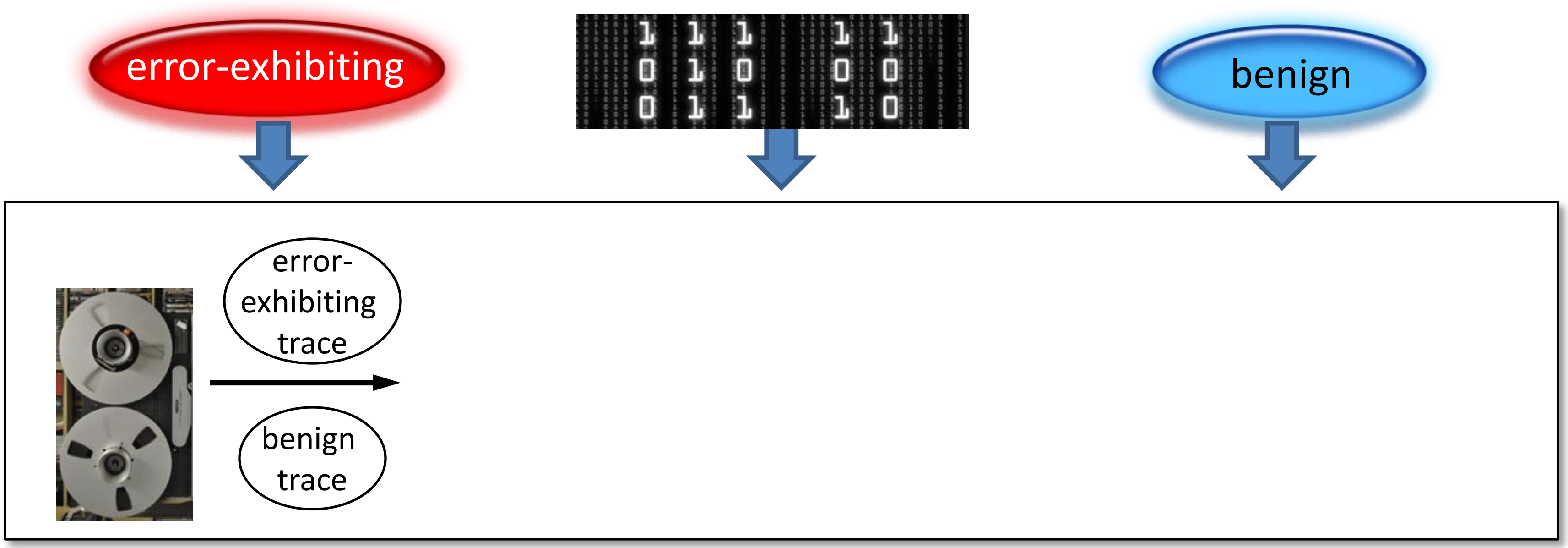


  - msvcr71.dll, hxds.dll on Windows

FlowStitch



FlowStitch


error-exhibiting
benign
error-exhibiting trace
benign trace

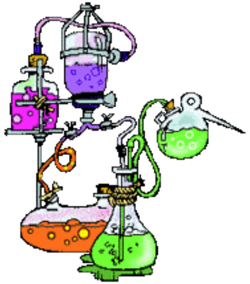
FlowStitch

error-exhibiting

benign

error-exhibiting trace

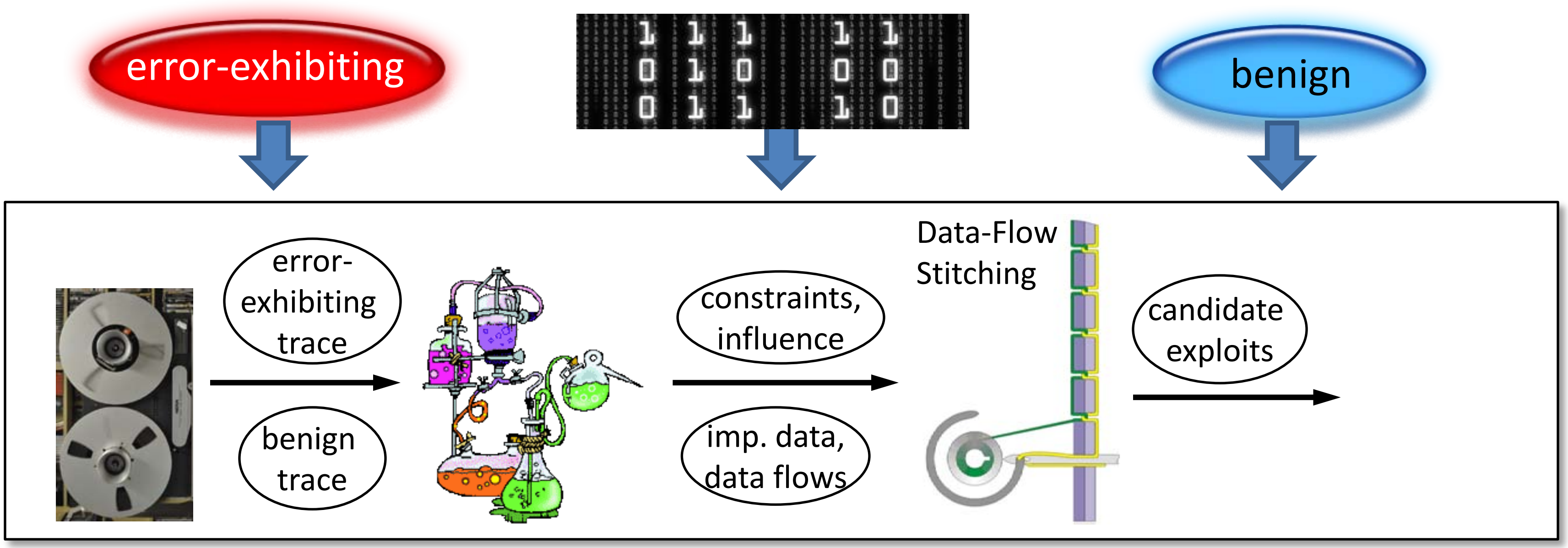
benign trace



constraints, influence

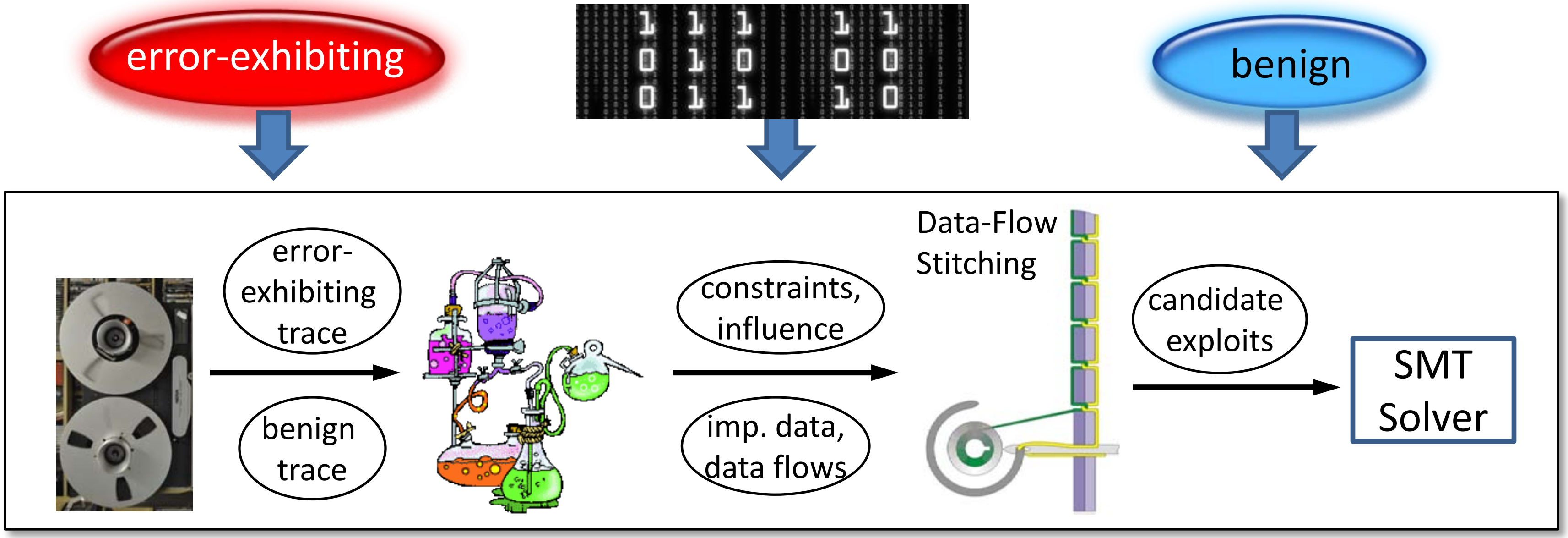
imp. data, data flows

FlowStitch

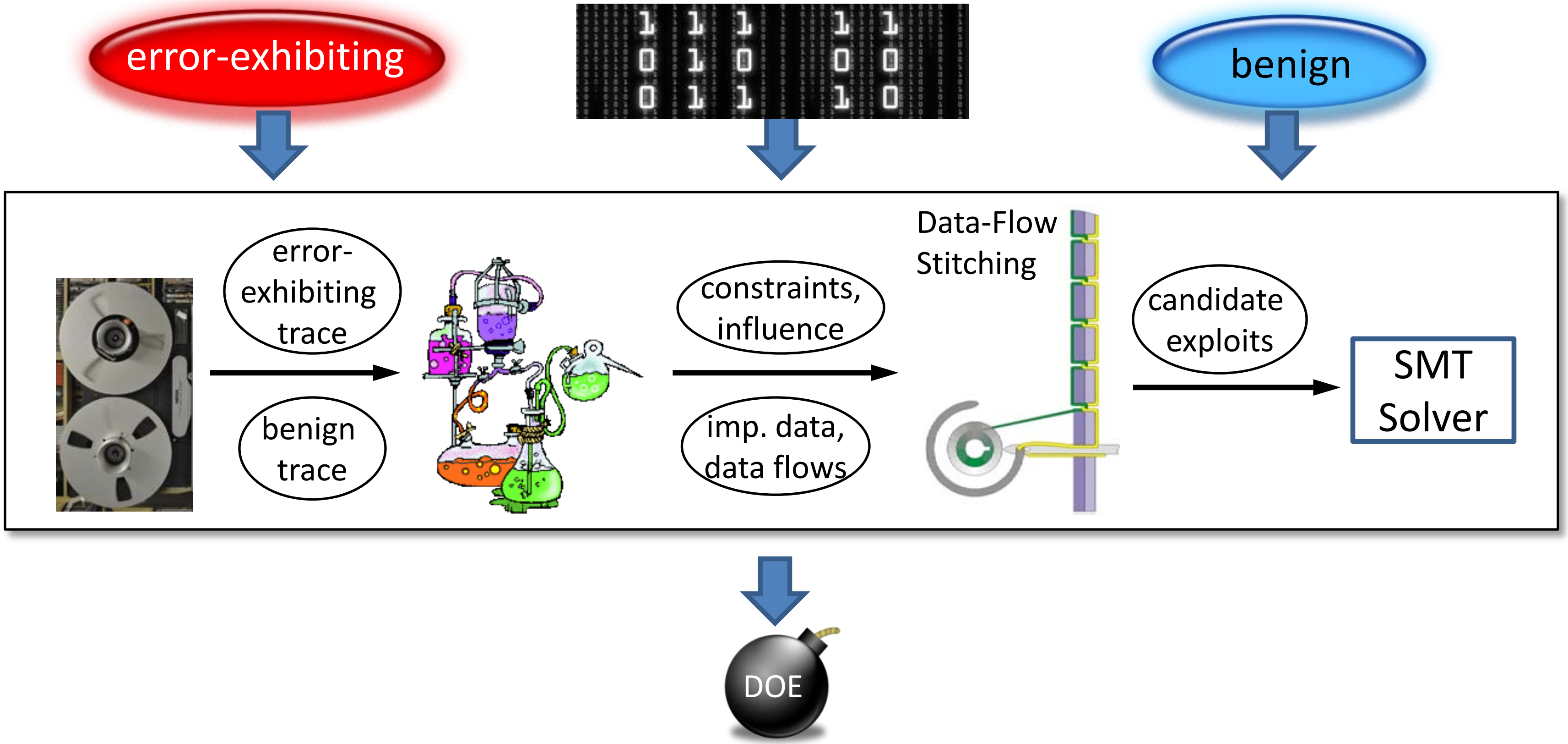


error-exhibiting
benign
error-exhibiting trace
benign trace
constraints, influence
imp. data, data flows
Data-Flow Stitching
candidate exploits

FlowStitch


error-exhibiting
1 1 1 1 1
0 1 0 0 0
0 1 1 1 0
benign
error-exhibiting trace
benign trace
constraints, influence
imp. data, data flows
Data-Flow Stitching
candidate exploits
SMT Solver

FlowStitch


error-exhibiting
1 1 1 1 1
0 1 0 0 0
0 1 1 1 0
benign
error-exhibiting trace
benign trace
constraints, influence
imp. data, data flows
Data-Flow Stitching
candidate exploits
SMT Solver
DOE

Evaluation --- Generated Exploits

ID	Vul. bin	Vulnerability	Data-Oriented Exploits	ASLR
CVE-2013-2028	nginx	Stack bof	L_0 : private key	
			M_0: http root dir	
CVE-2012-0809	sudo	Format string	M_0: user id	✓
CVE-2009-4769	httpdx	Format string	L_0: admin's passwd	✓
			M_0: admin;s passwd	✓
			M_1: anon.'s permission	✓
			M_2: anon.'s root dir	✓
			M_3: CGI root dir	✓
bugtraq ID: 41956	orzhttpd	Format string	L_0: randomized addr	✓
			M_0: http root dir	✓
CVE-2002-1496 *	nullhttpd	Heap overflow	M_0: http root dir	
			M_1: CGI root dir	
CVE-2001-0820 *	ghttpd	Stack bof	M_0: CGI root dir	
CVE-2001-0144 *	SSHD	integer overflow	L_0: root passwd hash	
			M_0: user id	
			M_1: authenticated flag	
CVE-2000-0573 *	wu-ftpd	Format string	L_0: env variables	
			M_0: user id (single-edge)	✓
			M_1: user id (pointer stitch)	✓

* CVEs discussed in Shuo Chen's work [1]

Evaluation --- Generated Exploits

ID	Vul. bin	Vulnerability	Data-Oriented Exploits	ASLR
CVE-2013-2028	nginx	Stack bof	L_0 : private key	
			M_0: http root dir	
CVE-2012-0809	sudo	Format string	M_0: user id	✓
CVE-2009-4769	httpdx	Format string	L_0: admin's passwd	✓
			M_0: admin;s passwd	✓
			M_1: anon.'s permission	✓
			M_2: anon.'s root dir	✓
			M_3: CGI root dir	✓
bugtraq ID: 41956	orzhttpd	Format string	L_0: randomized addr	✓
			M_0: http root dir	✓
CVE-2002-1496 *	nullhttpd	Heap overflow	M_0: http root dir	
			M_1: CGI root dir	
CVE-2001-0820 *	ghttpd	Stack bof	M_0: CGI root dir	
CVE-2001-0144 *	SSHD	integer overflow	L_0: root passwd hash	
			M_0: user id	
			M_1: authenticated flag	
CVE-2000-0573 *	wu-ftpd	Format string	L_0: env variables	
			M_0: user id (single-edge)	✓
			M_1: user id (pointer stitch)	✓

* CVEs discussed in Shuo Chen's work [1]

- 19 exploits

Evaluation --- Generated Exploits

ID	Vul. bin	Vulnerability	Data-Oriented Exploits	ASLR
CVE-2013-2028	nginx	Stack bof	L_0 : private key	
			M_0: http root dir	
CVE-2012-0809	sudo	Format string	M_0: user id	✓
CVE-2009-4769	httpdx	Format string	L_0: admin's passwd	✓
			M_0: admin;s passwd	✓
			M_1: anon.'s permission	✓
			M_2: anon.'s root dir	✓
			M_3: CGI root dir	✓
bugtraq ID: 41956	orzhttpd	Format string	L_0: randomized addr	✓
			M_0: http root dir	✓
CVE-2002-1496 *	nullhttpd	Heap overflow	M_0: http root dir	
			M_1: CGI root dir	
CVE-2001-0820 *	ghttpd	Stack bof	M_0: CGI root dir	
CVE-2001-0144 *	SSHD	integer overflow	L_0: root passwd hash	
			M_0: user id	
			M_1: authenticated flag	
CVE-2000-0573 *	wu-ftpd	Format string	L_0: env variables	
			M_0: user id (single-edge)	✓
			M_1: user id (pointer stitch)	✓

* CVEs discussed in Shuo Chen's work [1]

- 19 exploits

Evaluation --- Generated Exploits

ID	Vul. bin	Vulnerability	Data-Oriented Exploits	ASLR
CVE-2013-2028	nginx	Stack bof	L_0 : private key	
			M_0: http root dir	
CVE-2012-0809	sudo	Format string	M_0: user id	✓
CVE-2009-4769	httpdx	Format string	L_0: admin's passwd	✓
			M_0: admin;s passwd	✓
			M_1: anon.'s permission	✓
			M_2: anon.'s root dir	✓
			M_3: CGI root dir	✓
bugtraq ID: 41956	orzhttpd	Format string	L_0: randomized addr	✓
			M_0: http root dir	✓
CVE-2002-1496 *	nullhttpd	Heap overflow	M_0: http root dir	
			M_1: CGI root dir	
CVE-2001-0820 *	ghttpd	Stack bof	M_0: CGI root dir	
CVE-2001-0144 *	SSHD	integer overflow	L_0: root passwd hash	
			M_0: user id	
			M_1: authenticated flag	
CVE-2000-0573 *	wu-ftpd	Format string	L_0: env variables	
			M_0: user id (single-edge)	✓
			M_1: user id (pointer stitch)	✓

* CVEs discussed in Shuo Chen's work [1]

- 19 exploits
- 16 prev. unknown

Evaluation --- Generated Exploits

ID	Vul. bin	Vulnerability	Data-Oriented Exploits	ASLR
CVE-2013-2028	nginx	Stack bof	L_0 : private key	
			M_0: http root dir	
CVE-2012-0809	sudo	Format string	M_0: user id	✓
CVE-2009-4769	httpdx	Format string	L_0: admin's passwd	✓
			M_0: admin;s passwd	✓
			M_1: anon.'s permission	✓
			M_2: anon.'s root dir	✓
			M_3: CGI root dir	✓
bugtraq ID: 41956	orzhttpd	Format string	L_0: randomized addr	✓
			M_0: http root dir	✓
CVE-2002-1496 *	nullhttpd	Heap overflow	M_0: http root dir	
			M_1: CGI root dir	
CVE-2001-0820 *	ghttpd	Stack bof	M_0: CGI root dir	
CVE-2001-0144 *	SSHD	integer overflow	L_0: root passwd hash	
			M_0: user id	
			M_1: authenticated flag	
CVE-2000-0573 *	wu-ftpd	Format string	L_0: env variables	
			M_0: user id (single-edge)	✓
			M_1: user id (pointer stitch)	✓

* CVEs discussed in Shuo Chen's work [1]

- 19 exploits
- 16 prev. unknown
- 7 advanced stitch
  - 2-level stitch

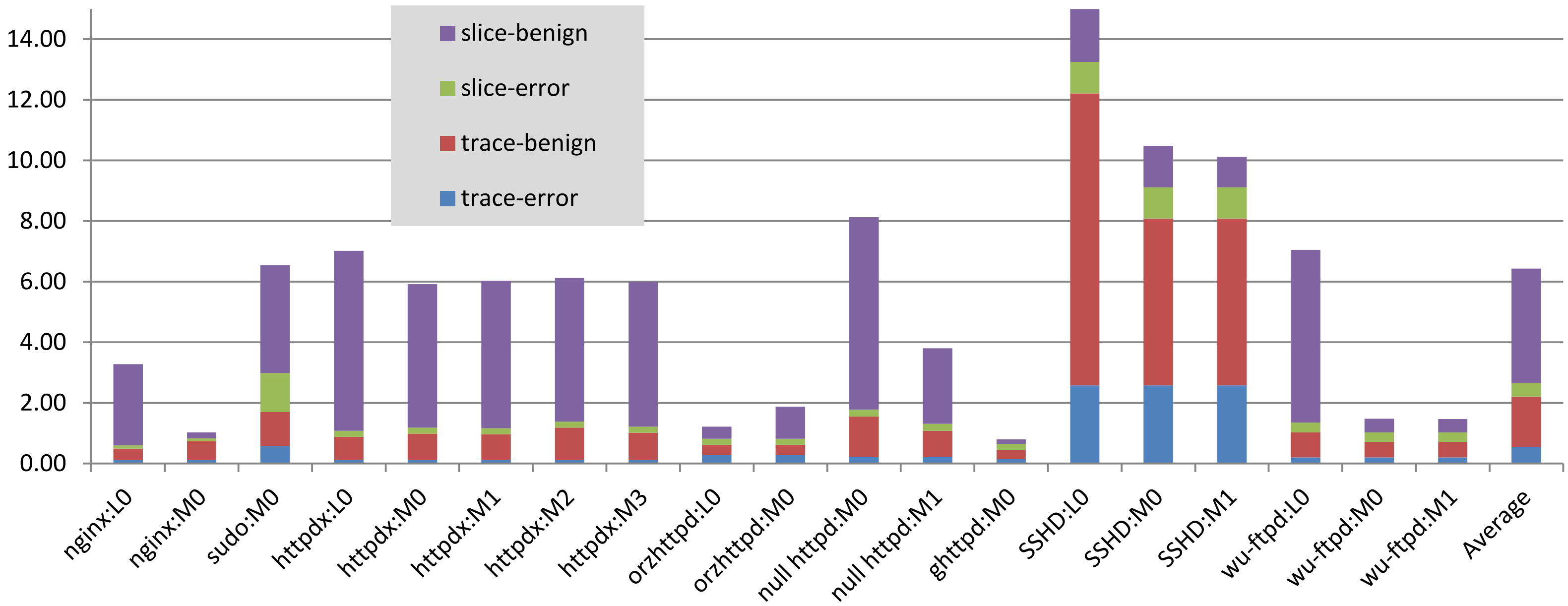
Evaluation --- Generated Exploits

ID	Vul. bin	Vulnerability	Data-Oriented Exploits	ASLR
CVE-2013-2028	nginx	Stack bof	L_0 : private key	
			M_0: http root dir	
CVE-2012-0809	sudo	Format string	M_0: user id	✓
CVE-2009-4769	httpdx	Format string	L_0: admin's passwd	✓
			M_0: admin;s passwd	✓
			M_1: anon.'s permission	✓
			M_2: anon.'s root dir	✓
			M_3: CGI root dir	✓
bugtraq ID: 41956	orzhttpd	Format string	L_0: randomized addr	✓
			M_0: http root dir	✓
CVE-2002-1496 *	nullhttpd	Heap overflow	M_0: http root dir	
			M_1: CGI root dir	
CVE-2001-0820 *	ghttpd	Stack bof	M_0: CGI root dir	
CVE-2001-0144 *	SSHD	integer overflow	L_0: root passwd hash	
			M_0: user id	
			M_1: authenticated flag	
CVE-2000-0573 *	wu-ftpd	Format string	L_0: env variables	
			M_0: user id (single-edge)	✓
			M_1: user id (pointer stitch)	✓

* CVEs discussed in Shuo Chen's work [1]

- 19 exploits
- 16 prev. unknown
- 7 advanced stitch
  - 2-level stitch
- 10 bypass ASLR
  - 8 fixed addresses
  - 2 address reuse

Evaluation --- Performance



- 6.5 min/exploit
- Slice takes long
  - faster version is available (binary version)

Case Study – 2-Level Stitch

- *ghttpd* web server: stack buffer overflow

//serveconnection():
char *ptr; //URL pointer
//esi is allocated for it
1: if(strstr(ptr,"/.."))
reject the request;
2: **log**(...);
3: exec(ptr);
	Assembly of log(...)
	push %ebp
	push %esi
	// stack overflow
	pop %esi
	pop %ebp
	ret
		Assembly of line 3:
		
		push %esi
		…
		call <exec@plt>

- Previous exploit[1]
  - Corrupt pointer ptr: *(ptr) -> url

Case Study – 2-Level Stitch

- *ghttpd* web server: stack buffer overflow

```
   //serveconnection():
   char *ptr; //URL pointer
   //esi is allocated for it
1: if(strstr(ptr,"/.."))
      reject the request;
2: log(...);
3: exec(ptr);
```

```
Assembly of log(...)
  push %ebp
  push %esi
  // stack overflow
  pop %esi
  pop %ebp
  ret
```

```
Assembly of line 3:
  mov  -0xc(%ebp), %esi
  push %esi
  …
  call <exec@plt>
```

- Previous exploit[1] does not work any more
  - Corrupt pointer ptr: *(ptr) -> url

Case Study – 2-Level Stitch

- *ghttpd* web server: stack buffer overflow

`//serveconnection():`
`char *ptr; //URL pointer`
`//esi is allocated for it`
`1: if(strstr(ptr,"/.."))`
`reject the request;`
`2: log(...);`
`3: exec(ptr);`
	`Assembly of log(...)`
	`push %ebp`
	`push %esi`
	`// stack overflow`
	`pop %esi`
	`pop %ebp`
	`ret`
		`Assembly of line 3:`
		`mov -0xc(%ebp), %esi`
		`push %esi`
		`…`
		`call <exec@plt>`

- Previous exploit[1] does not work any more
  - Corrupt pointer ptr: *(ptr) -> url
- We build a 2-level stitch
  - Corrupt pointer saved ebp: *(*(saved ebp)) -> *ptr -> url

Case Study – Sensitive Data Lifespan

- *SSHD* hashed key info leak
- *getspnam()* in *glibc* gets hashed key (heap copy) 
- *SSHD* copies hashed key to local stack (stack copy) 

Case Study – Sensitive Data Lifespan

- *SSHD* hashed key info leak
- *getspnam()* in *glibc* gets hashed key (heap copy)

- *SSHD* copies hashed key to local stack (stack copy)
  - Overwritten by later usage

Case Study – Sensitive Data Lifespan

- *SSHD* hashed key info leak
- *getspnam()* in *glibc* gets hashed key (heap copy)
  - *endspent()* in *glibc* releases memory, not clears it!
  - Still alive for stitching

- *SSHD* copies hashed key to local stack (stack copy)
  - Overwritten by later usage

Case Study – Sensitive Data Lifespan

- *SSHD* hashed key info leak
- *getspnam()* in *glibc* gets hashed key (heap copy)

  - *endspent()* in *glibc* releases memory, not clears it!
  - Still alive for stitching
- *SSHD* copies hashed key to local stack (stack copy)

  - Overwritten by later usage
- *Challenging* to make lifespan correct!

Conclusion

- Rich Category: Data-Oriented Exploits
  - Single-edge stitch, Pointer stitch
  - N-level stitch, Multi-flow stitch
- Data Flow Stitching
  - Systematic way to generate data-oriented exploits
  - Agnostic to CFI, DEP and often ASLR
- Automatic construction is feasible

Thanks!

Hong Hu

huhong@comp.nus.edu.sg

http://www.comp.nus.edu.sg/~huhong/