ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

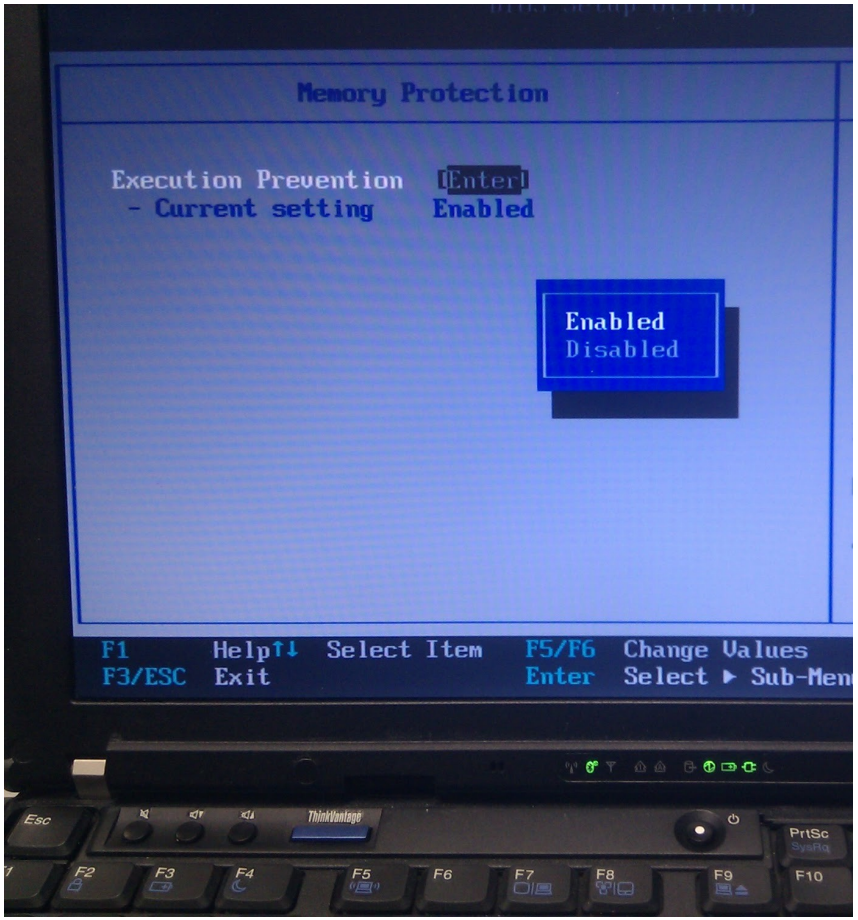
String Oriented Programming

Circumventing ASLR, DEP, and other Guards

Mathias Payer, ETH Zürich

Motivation

Additional protection mechanisms prevent many existing attack vectors



feature	8.04 LTS (Hardy Heron)	10.04 LTS (Lucid Lynx)	10.10 (Maverick Meerkat)	11.04 (Natty Narwhal)	11.10 (Oneiric Ocelot)	12.04 LTS (Precise Pangolin)
No Open Ports	policy	policy	policy	policy	policy	policy
Password hashing	md5	sha512	sha512	sha512	sha512	sha512
SYN cookies	–	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl	kernel & sysctl
Filesystem Capabilities	–	kernel	kernel	kernel	kernel	kernel
Configurable Firewall	ufw	ufw	ufw	ufw	ufw	ufw
PR_SET_SECCOMP	kernel	kernel	kernel	kernel	kernel	kernel
AppArmor	2.1	2.5	2.5.1	2.5.1	2.5.1	2.5.1
SELinux	universe	universe	universe	universe	universe	universe
SMACK	–	kernel	kernel	kernel	kernel	kernel
Encrypted LVM	alt installer	alt installer	alt installer	alt installer	alt installer	alt installer
eCryptfs	–	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames	~/Private or ~, filenames
Stack Protector	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Heap Protector	glibc	glibc	glibc	glibc	glibc	glibc
Pointer Obfuscation	glibc	glibc	glibc	glibc	glibc	glibc
Stack ASLR	kernel	kernel	kernel	kernel	kernel	kernel
Libs/mmap ASLR	kernel	kernel	kernel	kernel	kernel	kernel
Exec ASLR	kernel (-mm patch)	kernel	kernel	kernel	kernel	kernel
brk ASLR	kernel (exec ASLR)	kernel	kernel	kernel	kernel	kernel
VDSO ASLR	kernel	kernel	kernel	kernel	kernel	kernel
Built as PIE	–	package list	package list	package list	package list	package list
Built with Fortify Source	–	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Built with RELRO	–	gcc patch	gcc patch	gcc patch	gcc patch	gcc patch
Built with BIND_NOW	–	package list	package list	package list	package list	package list
Non-Executable Memory	PAE only	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation	PAE, ia32 partial-NX-emulation
/proc/$pid/maps protection	kernel & sysctl	kernel	kernel	kernel	kernel	kernel
Symlink restrictions	–	–	kernel	kernel	kernel	kernel
Hardlink restrictions	–	–	kernel	kernel	kernel	kernel
ptrace scope	–	–	kernel	kernel	kernel	kernel
0-address protection	kernel & sysctl	kernel	kernel	kernel	kernel	kernel
/dev/mem protection	kernel (-mm patch)	kernel	kernel	kernel	kernel	kernel
/dev/kmem disabled	kernel (-mm patch)	kernel	kernel	kernel	kernel	kernel
Block module loading	drop CAP_SYS_MODULES	sysctl	sysctl	sysctl	sysctl	sysctl
Read-only data sections	kernel	kernel	kernel	kernel	kernel	kernel
Stack protector	–	kernel	kernel	kernel	kernel	kernel
Module RO/NX	–	–	–	kernel	kernel	kernel
Kernel Address Display Restriction	–	–	–	kernel	kernel	kernel
Blacklist Rare Protocols	–	–	–	kernel	kernel	kernel
Syscall Filtering	–	–	–	–	kernel	kernel

Motivation

Additional protection mechanisms prevent many existing attack vectors

Format string exploits are often overlooked

- Drawback: hard to construct (new protection mechanisms)
- Define a way to deterministically exploit format string bugs

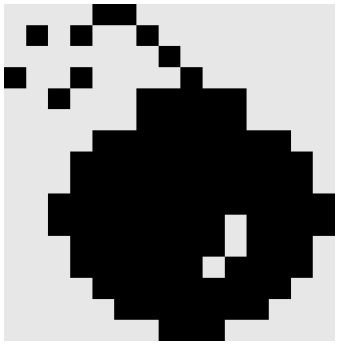
Attack model

Attacker with restricted privileges forces escalation

Attacker knows source code and binary

Successful attacks

- Redirect control flow to alternate location
- Injected code is executed or alternate data is used for existing code



Outline

Motivation

Attack model

Attack vectors and protection mechanisms

String Oriented Programming

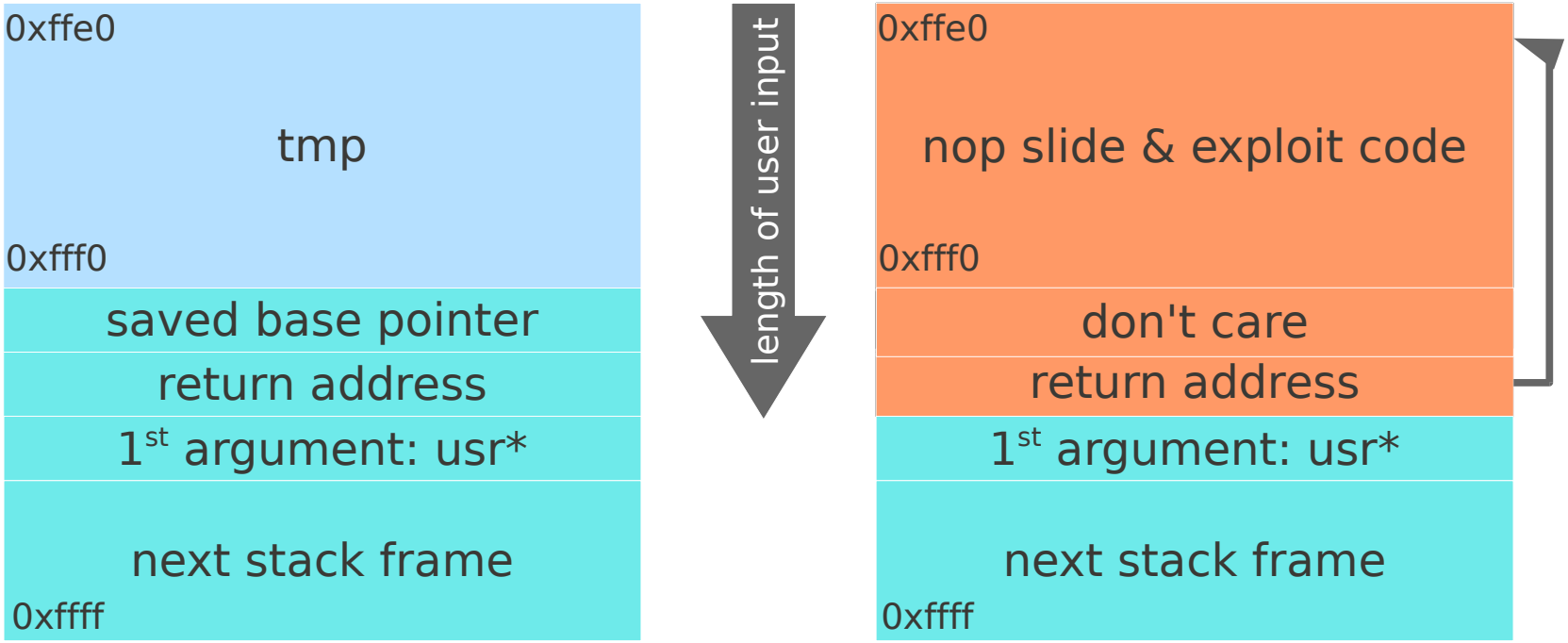
Conclusion

Code injection*

Injects additional code into the runtime image

- Buffer overflow used to inject code as data

```
void foo(char *usr)
{
  char tmp[len];
  strcpy(tmp, usr);
}
```



* Aleph1, Phrack #49

Code injection*

Injects additional code into the runtime image

- Buffer overflow used to inject code as data

```
void foo(char *usr)
{
  char tmp[len];
  strcpy(tmp, usr);
}
```

Modern hardware and operating systems separate data and code

- Code injection is no longer feasible due to W ⊕ X
- If the attacked program uses a JIT then WX pages might be available

* Aleph1, Phrack #49

Protection mechanisms

Data Execution Prevention (DEP / ExecShield)

- Enforces the executable bit (W ⊕ X) on page granularity
- Changes: HW, kernel, loader

Address Space Layout Randomization (ASLR)

- All memory addresses (heap / stack / libraries) are dynamic
- Application itself is static
- Changes: loader

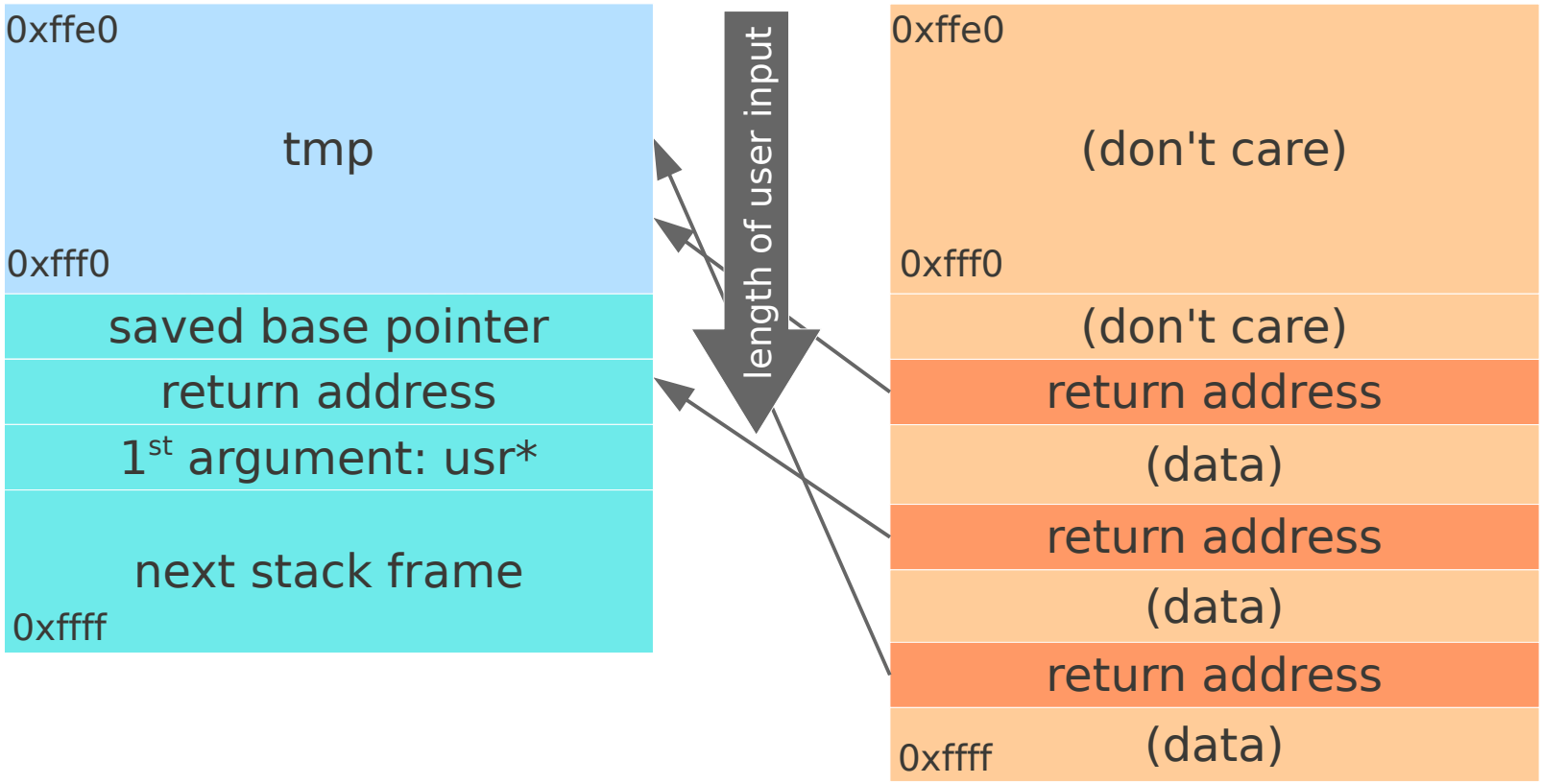
ProPolice (in gcc)

- Uses canaries on the stack to protect from stack-based overflows
- Changes: compiler

Return Oriented Programming (ROP)*

ROP prepares several stack invocation frames

- Executes arbitrary code
- Stack-based buffer overflow as initial attack vector



* Shacham, CCS'07

Return Oriented Programming (ROP)*

ROP prepares several stack invocation frames

- Executes arbitrary code
- Stack-based buffer overflow as initial attack vector

Executes alternate data with existing code

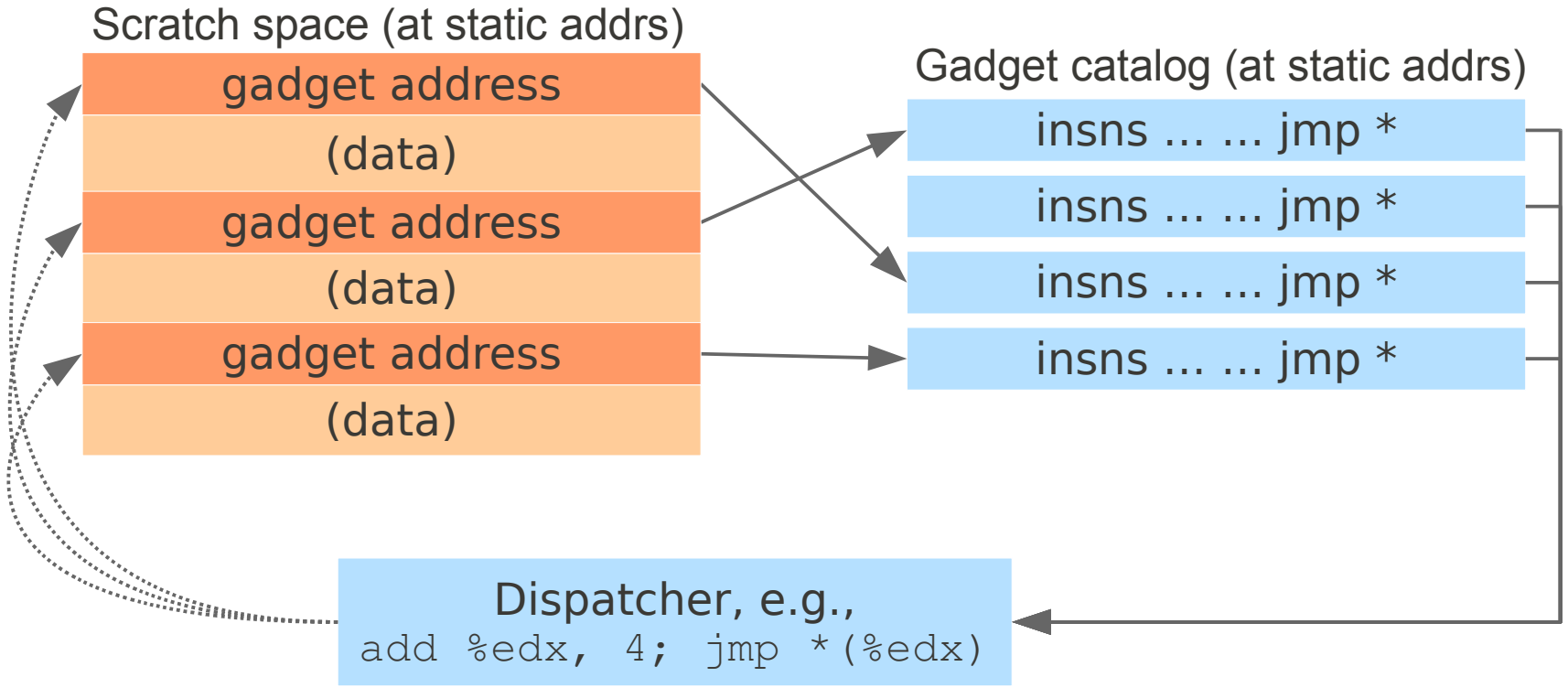
- Circumvents W ⊕ X
- Hard to get around ASLR, ProPolice

* Shacham, CCS'07

Jump Oriented Programming (JOP)*

Uses dispatchers and indirect control flow transfers

- JOP extends and generalizes ROP
- Any data region can be used as scratch space



* Bletsch et al., ASIACCS'11

Jump Oriented Programming (JOP)*

Uses dispatchers and indirect control flow transfers

- JOP extends and generalizes ROP
- Any data region can be used as scratch space

Executes alternate data with existing code

- Circumvents W ⊕ X
- Hard to get around ASLR, ProPolice (if stack data used)

* Bletsch et al., ASIACCS'11

Format string attack*

Attacker controlled format results in random writes

- Format strings consume parameters on the stack
- %n token inverses order of input, results in indirect memory write
- Often string is on stack and can be used to store pointers

Write 0xc0f3babe to 0x41414141:

```
printf(
  "AAAACAAA"          /* encode 2 halfword pointers */
  "%1$49387c"         /* write 0xc0f3 - 8 bytes */
  "%6$hn"             /* store at second HW */
  "%1$63947c%5$hn"  /* repeat with 0xbabe */
);
```

* many, e.g., Haas, Defcon 18

Format string attack*

Attacker controlled format results in random writes

- Format strings consume parameters on the stack
- %n token inverses order of input, results in indirect memory write
- Often string is on stack and can be used to store pointers

Write 0xc0f3babe to 0x41414141:

- `printf("AAAACAAA%1$49387c%6$hn%1$63947c%5$hn");`

Random writes are used to:

- Redirect control flow
- Prepare/inject malicious data

* many, e.g., Haas, Defcon 18

Outline

Motivation

Attack model

Attack vectors and protection mechanisms

String Oriented Programming

Conclusion

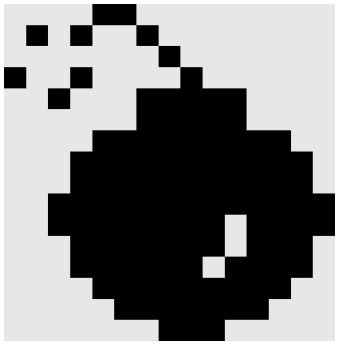
String Oriented Programming (SOP)

SOP executes arbitrary code (through data)

- Needed: format string bug, attacker-controlled buffer on stack
- Not needed: buffer overflow, executable memory regions

Executing code

- SOP builds on ROP/JOP
- Overwrites static instruction pointers (to initial ROP/JOP gadgets)



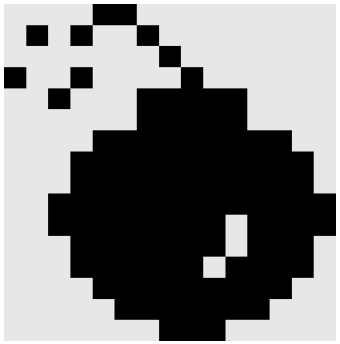
String Oriented Programming

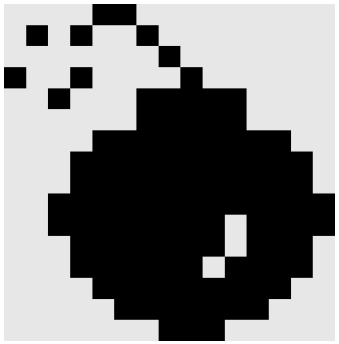
SOP patches and resolves addresses

- Application is static (this includes application's .plt and .got)
- Static program locations used to resolve relative addresses

Resolving hidden functions

- ASLR randomizes ~10bit for libraries
- Modify parts of static .got pointers
- Hidden functions can be called without loader support





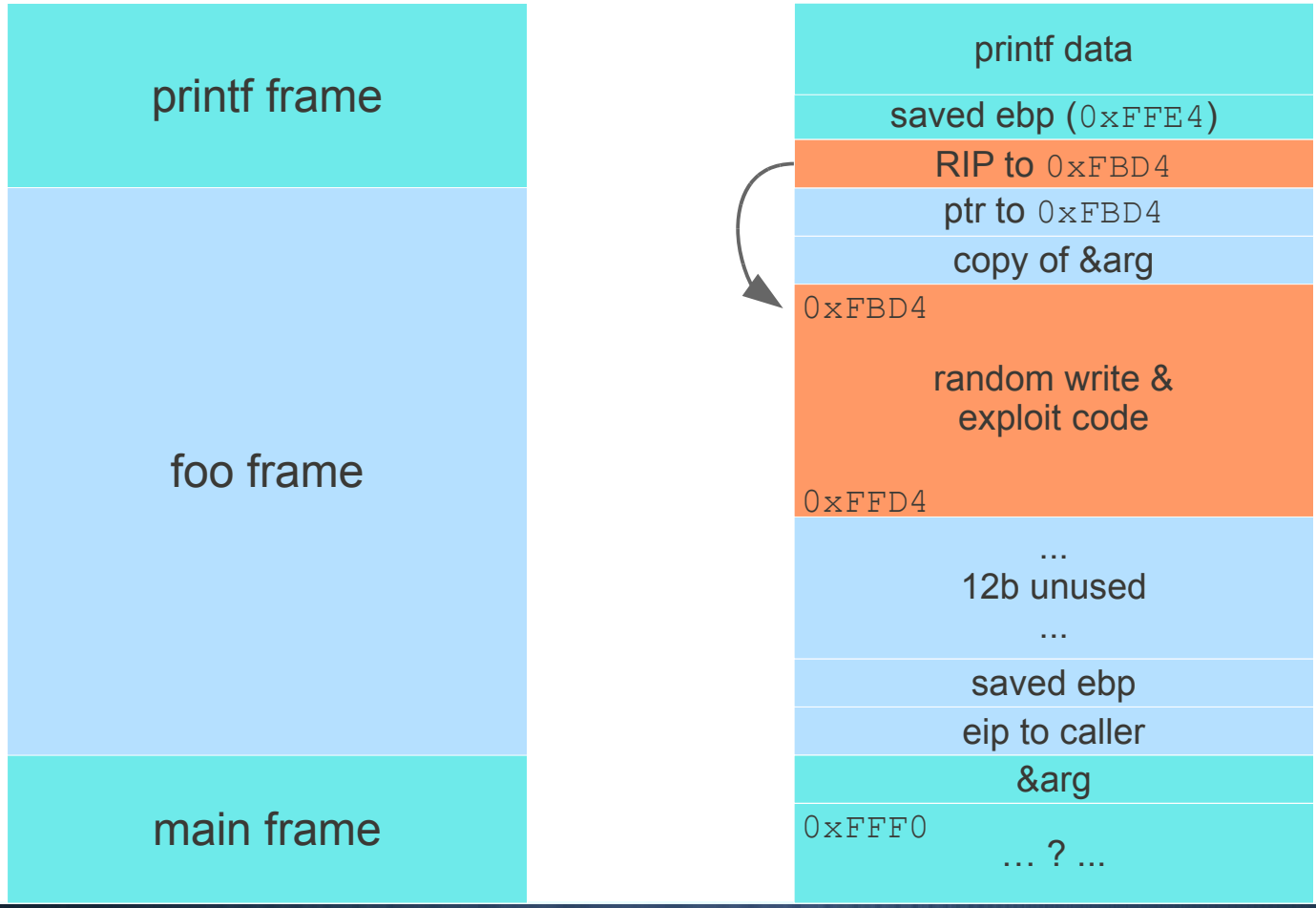
Running example

```
void foo(char *arg) {
  char text[1024];              // buffer on stack
  if (strlen(arg) >= 1024)      // length check
    return;
  strcpy(text, arg);
  printf(text);                 // vulnerable printf
}

…

foo(user_str);                  // unchecked user data

…
```

SOP: No Protection

All addresses are known, no execution protection, no stack protection

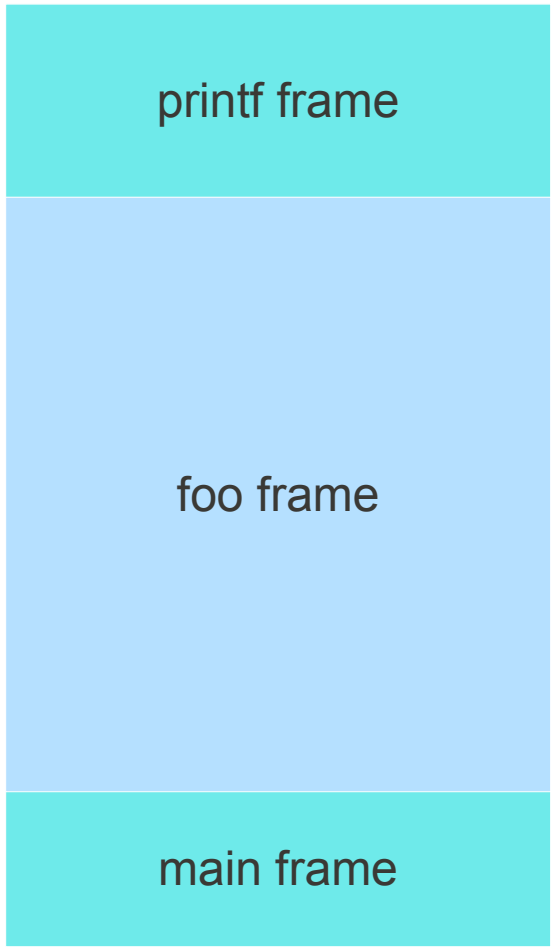
- Redirects control flow to code in the format string itself

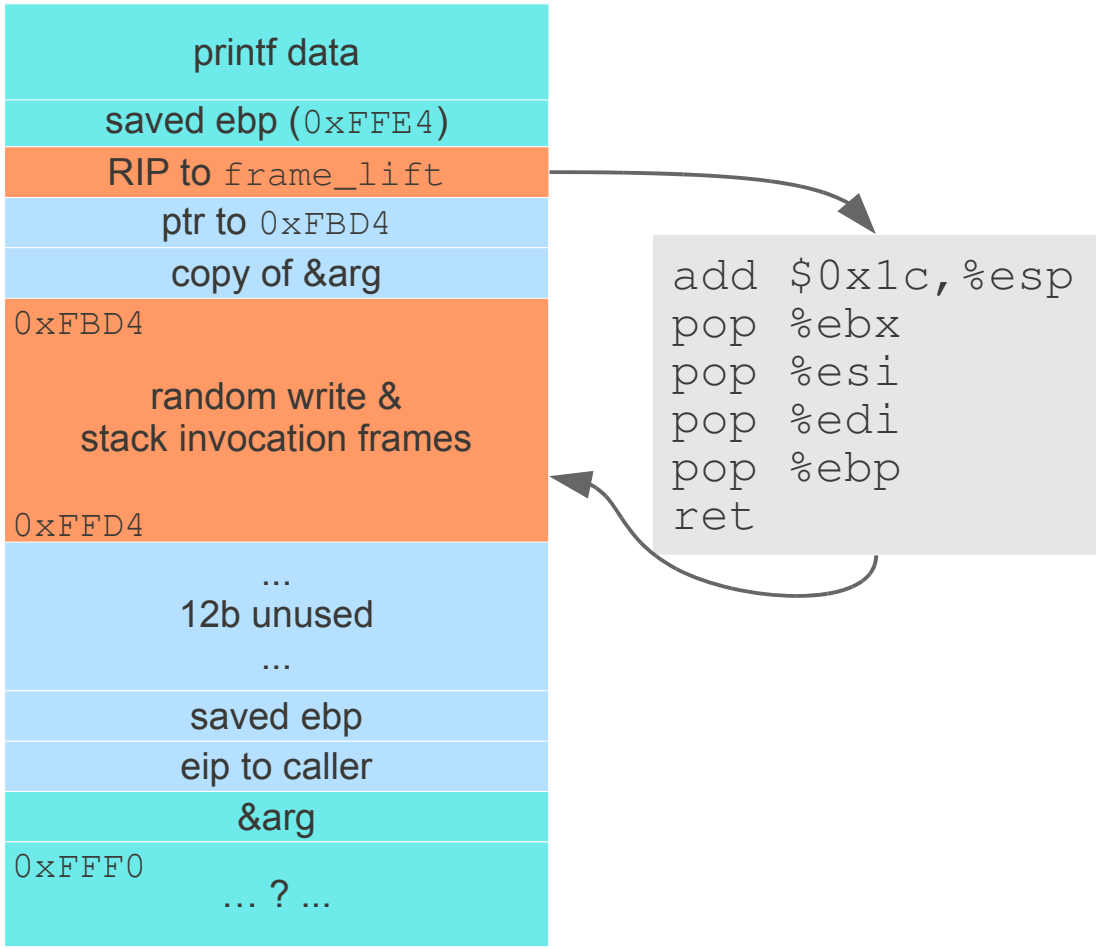


SOP: Only DEP

DEP prevents code injection, rely on ROP/JOP instead

GNU C compiler adds `frame_lift` gadget

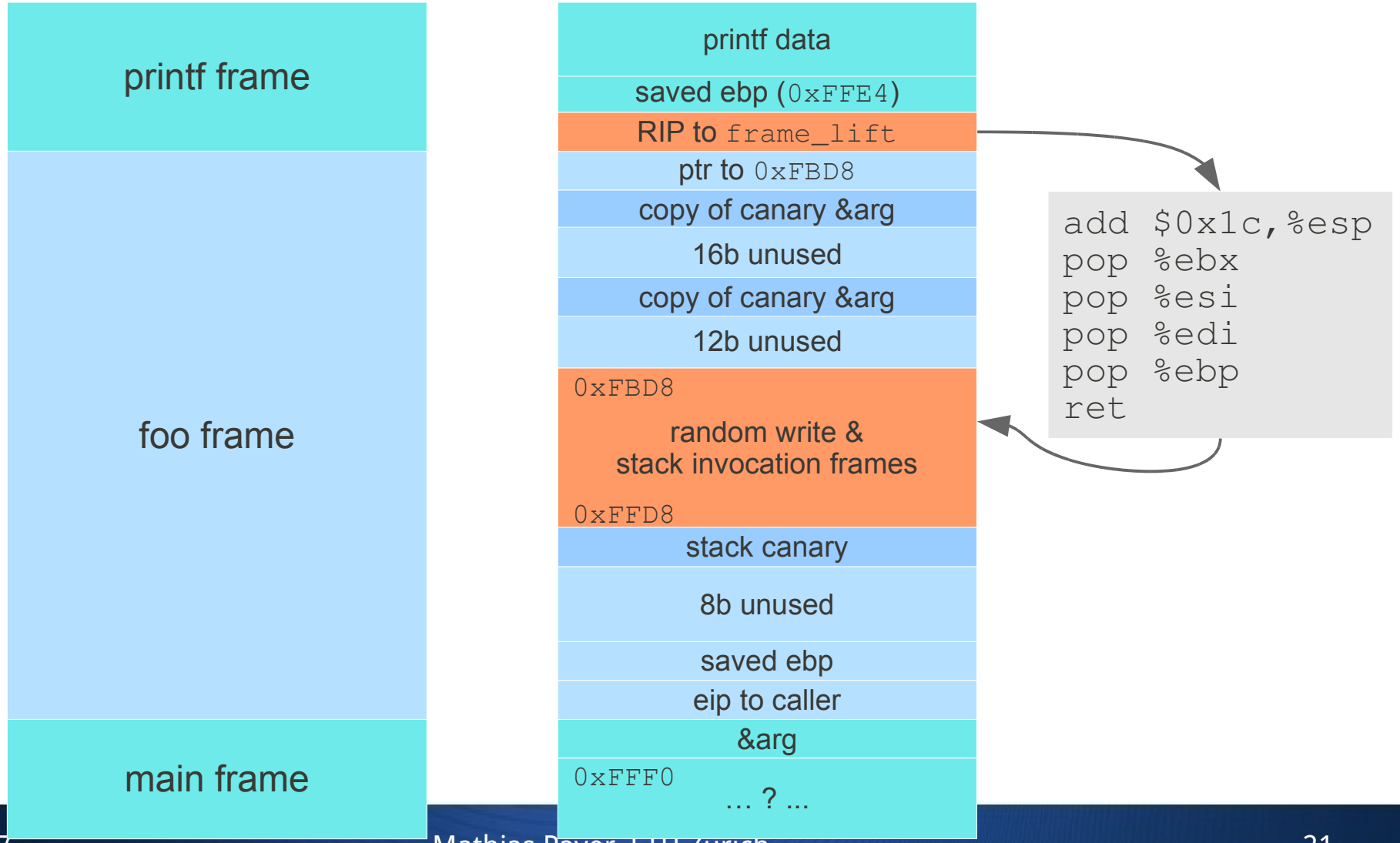




SOP: DEP & ProPolice

ProPolice uses/enforces stack canaries

- Reuse attack mechanism, keep canaries intact



SOP: ASLR, DEP, ProPolice

Combined defenses force SOP to reuse existing code

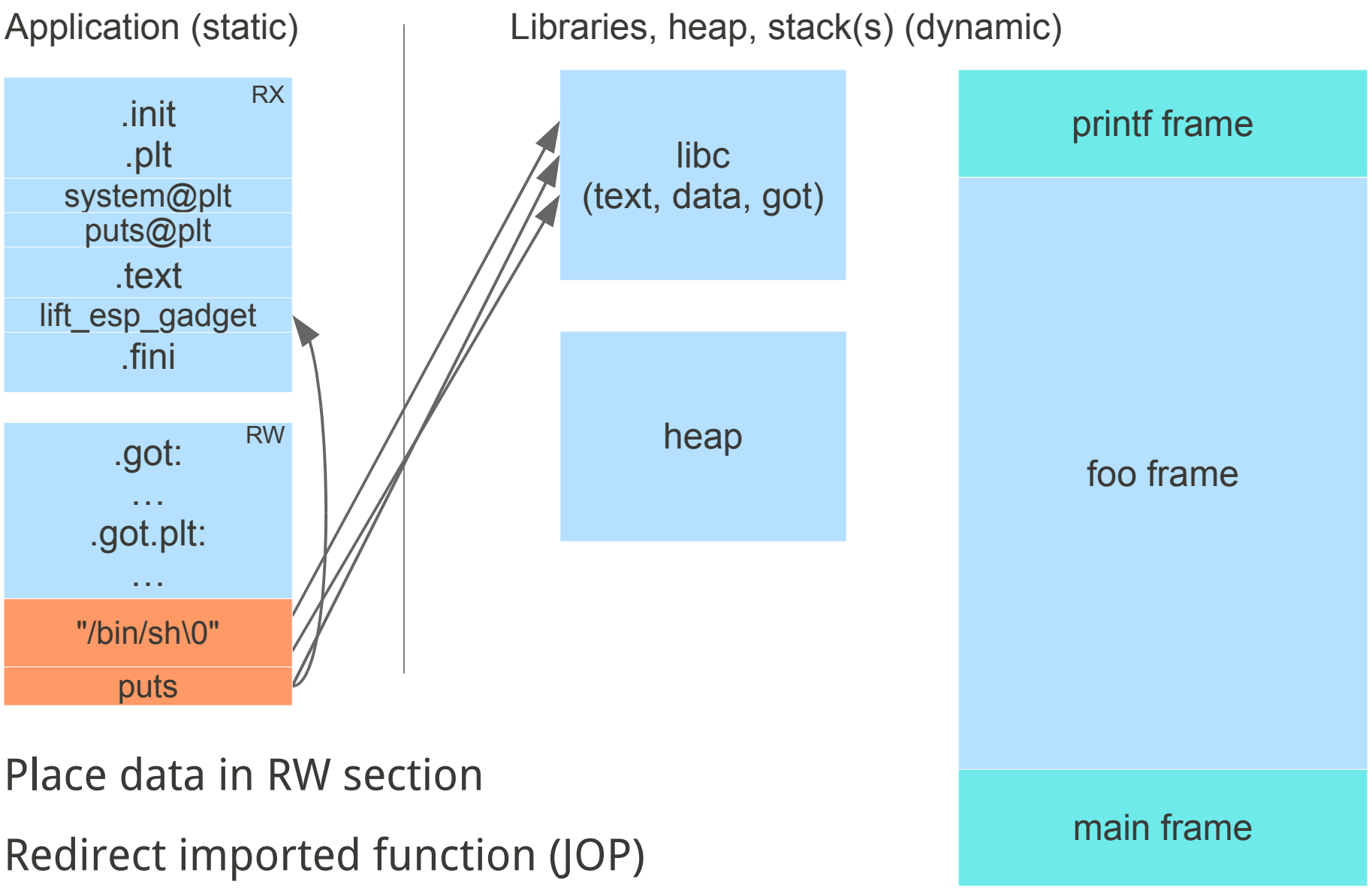
- Static code sequences in the application object
- Imported functions in the application (`.plt` and `.got`)

Use random byte-writes to adjust `.got` entries

- Enable other functions / gadgets that are not imported
- Combine stack invocation frames and indirect jump/call gadgets

```
void foo(char *prn)
{
  char text[1000];      // protected on stack
  strcpy(text, prn);
  printf(text);         // vulnerable printf
  puts("logged in\n"); // 'some' function
}
```

SOP: ASLR, DEP, ProPolice



Place data in RW section

Redirect imported function (JOP)

Use ROP for fun & profit

Outline

Motivation

Attack model

Attack vectors and protection mechanisms

String Oriented Programming

Conclusion

Conclusion

String Oriented Programming (SOP)

- Relies on format string exploit
- Extends data oriented programming (ROP / JOP)
- Naturally circumvents DEP and ProPolice
- Reconstructs pointers and circumvents ASLR

Format string bugs result in complete compromise of the application and full control for the attacker

- Protection against SOP needs more work (virtualization?)
- Look at the complete toolchain

Other protection mechanisms

Stack integrity (StackGuard, Propolice)

Verify library usage (Libsafe / Libverify)

Pointer encryption (PointGuard)

ISA modifications (ISA randomization)

Format string protection (FormatGuard)

Randomize memory locations (ASLR)

Check/verify control flow transfer (CFI / XFI)