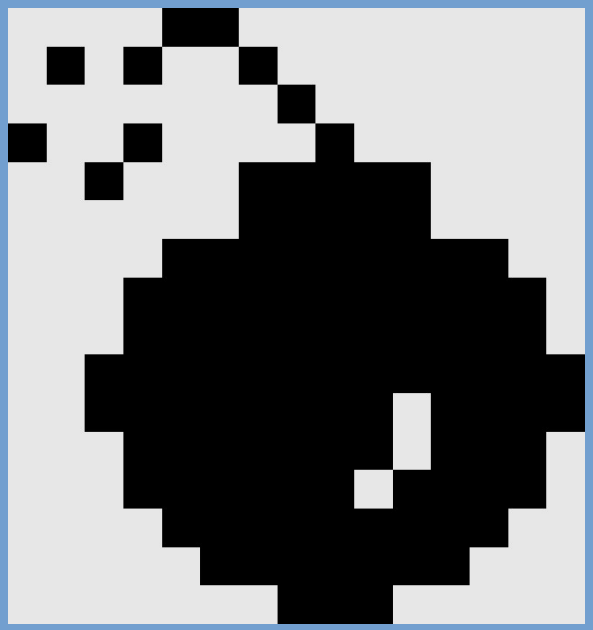
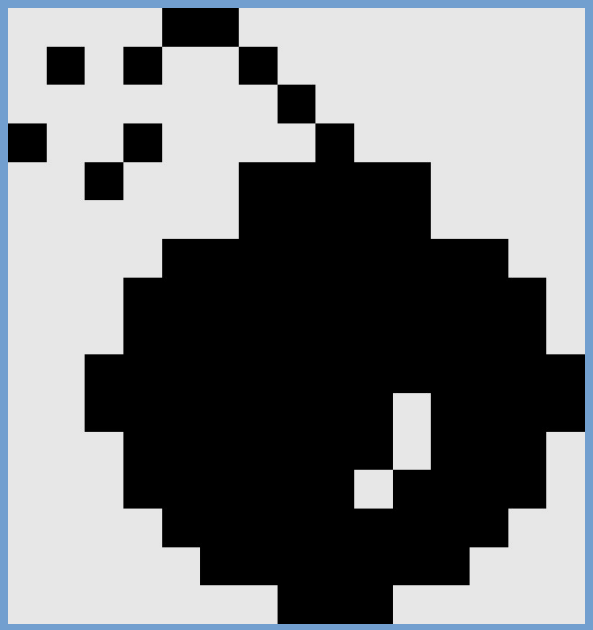
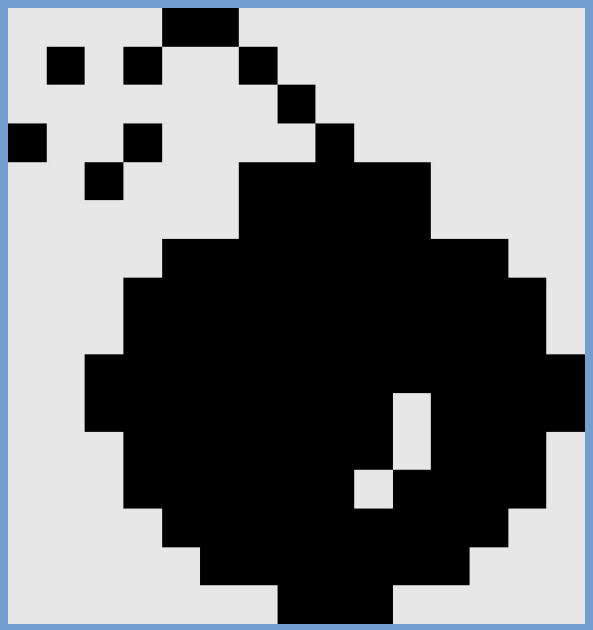




String Oriented Programming
Exploring Format String Attacks

Mathias Payer

hexhive

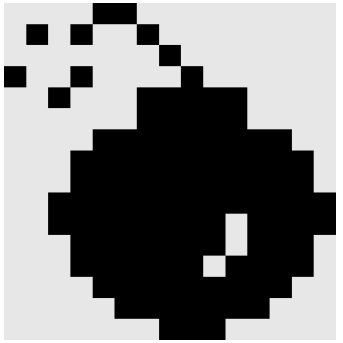
PURDUE UNIVERSITY

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

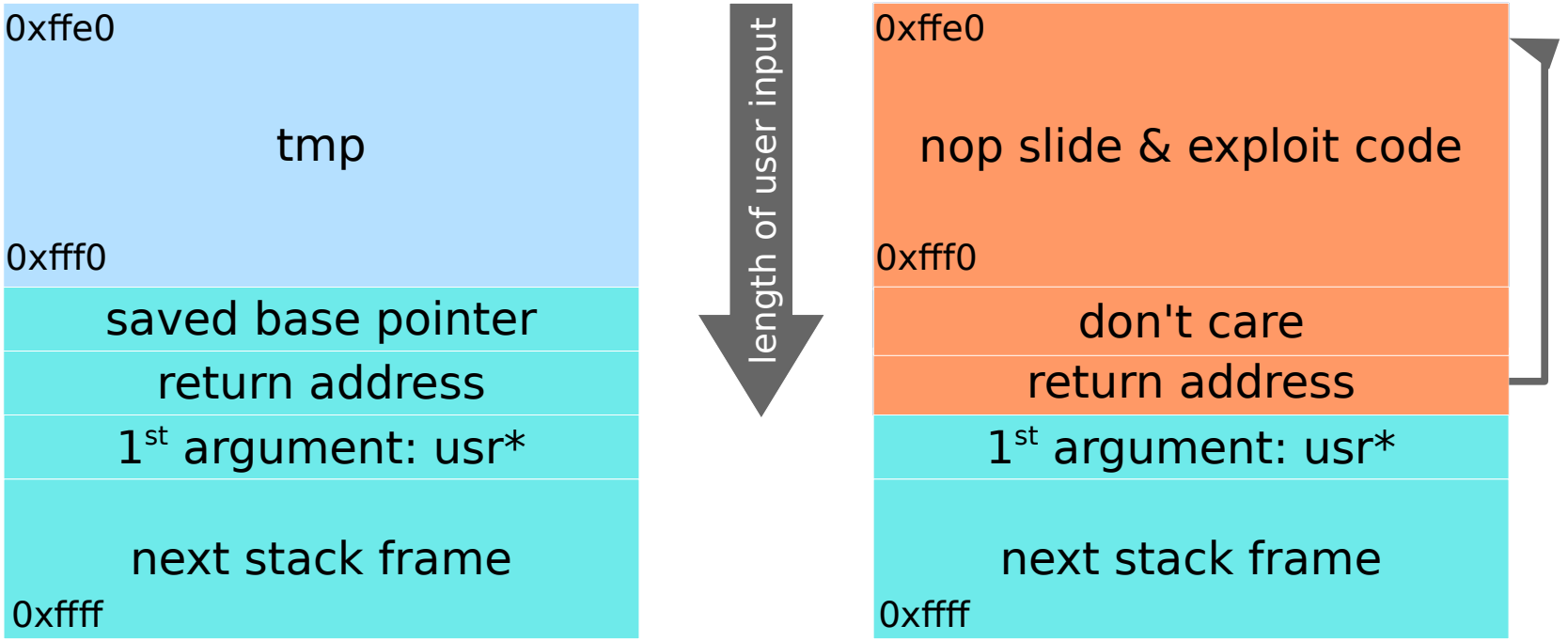
Outline

- 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

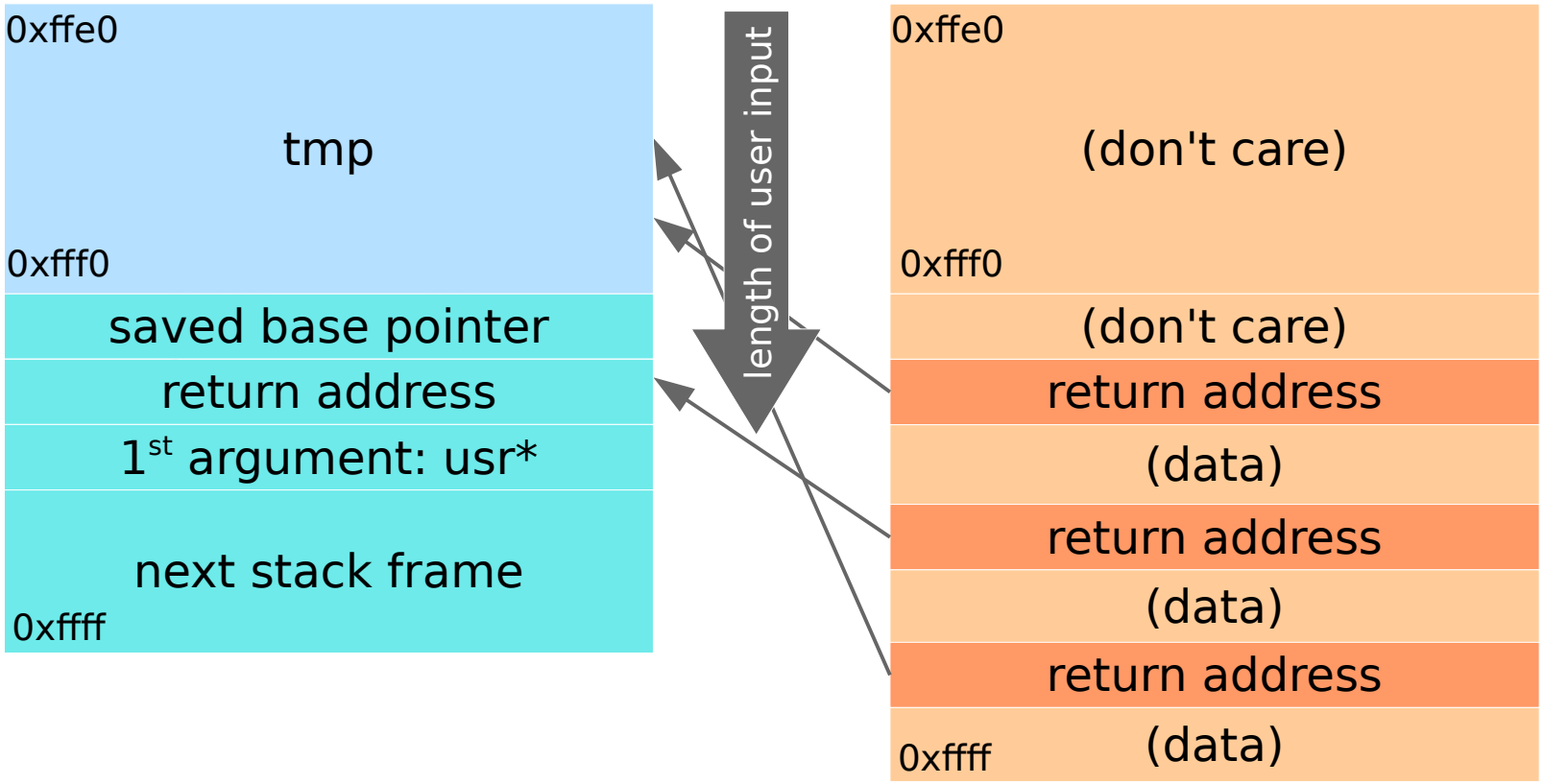
* Aleph1, Phrack #49

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

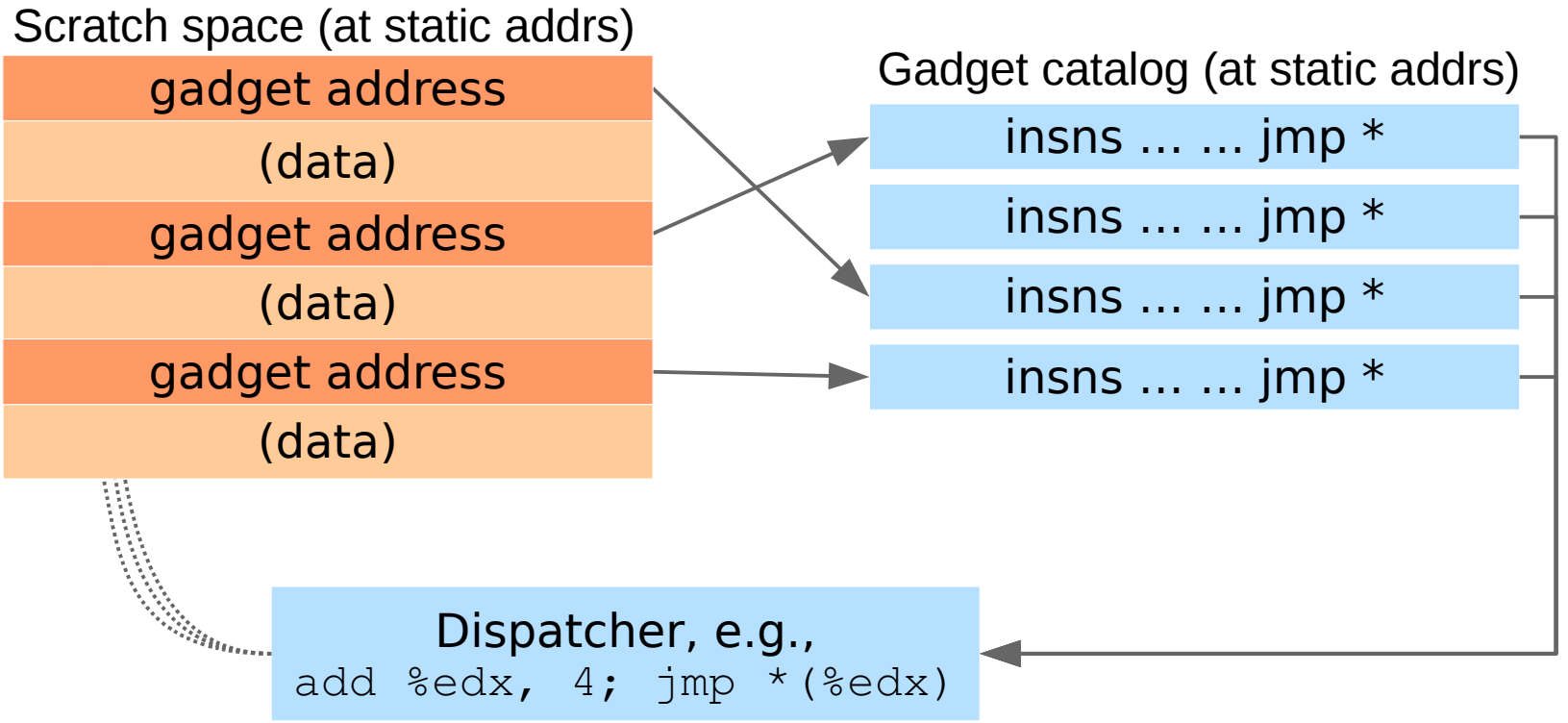
Return Oriented Programming (ROP)*

- 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

```
printf(
  "AAAACAAA"          /* encode 2 halfword pointers */
  "%1$49387c"         /* write 0xc0f3 - 8 bytes */
  "%6$hn"             /* store at second HW */
  "%1$61204c%5$hn" /* repeat with 0xb007 */
);
```

- Random writes are used to:
  - Redirect control flow
  - Prepare/inject malicious data

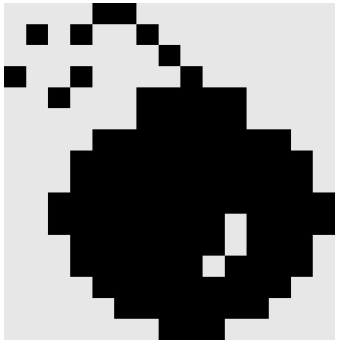
* many, e.g., Haas, Defcon 18

Outline

- Motivation
- Attack model
- Attack vectors and protection mechanisms
- String Oriented Programming
- Conclusion

String Oriented Programming (SOP)

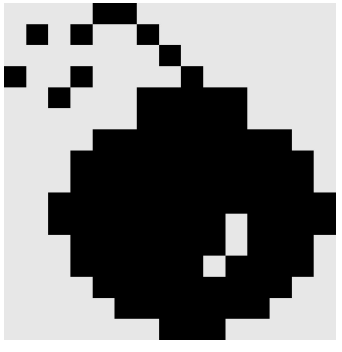
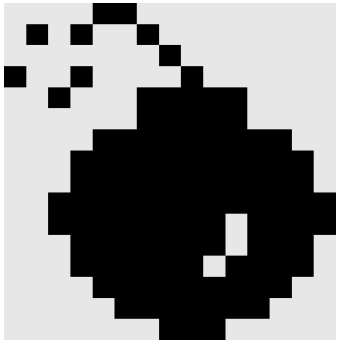
- SOP executes arbitrary code (through data)
  - Needed: format string bug, attacker-controlled buffer on stack
  - Not needed: buffer overflow, executable memory regions
- Executing code
  - SOP builds on ROP/JOP
  - Overwrites static instruction pointers (to initial ROP/JOP gadgets)



String Oriented Programming

- SOP patches and resolves addresses
  - Application is static (this includes .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

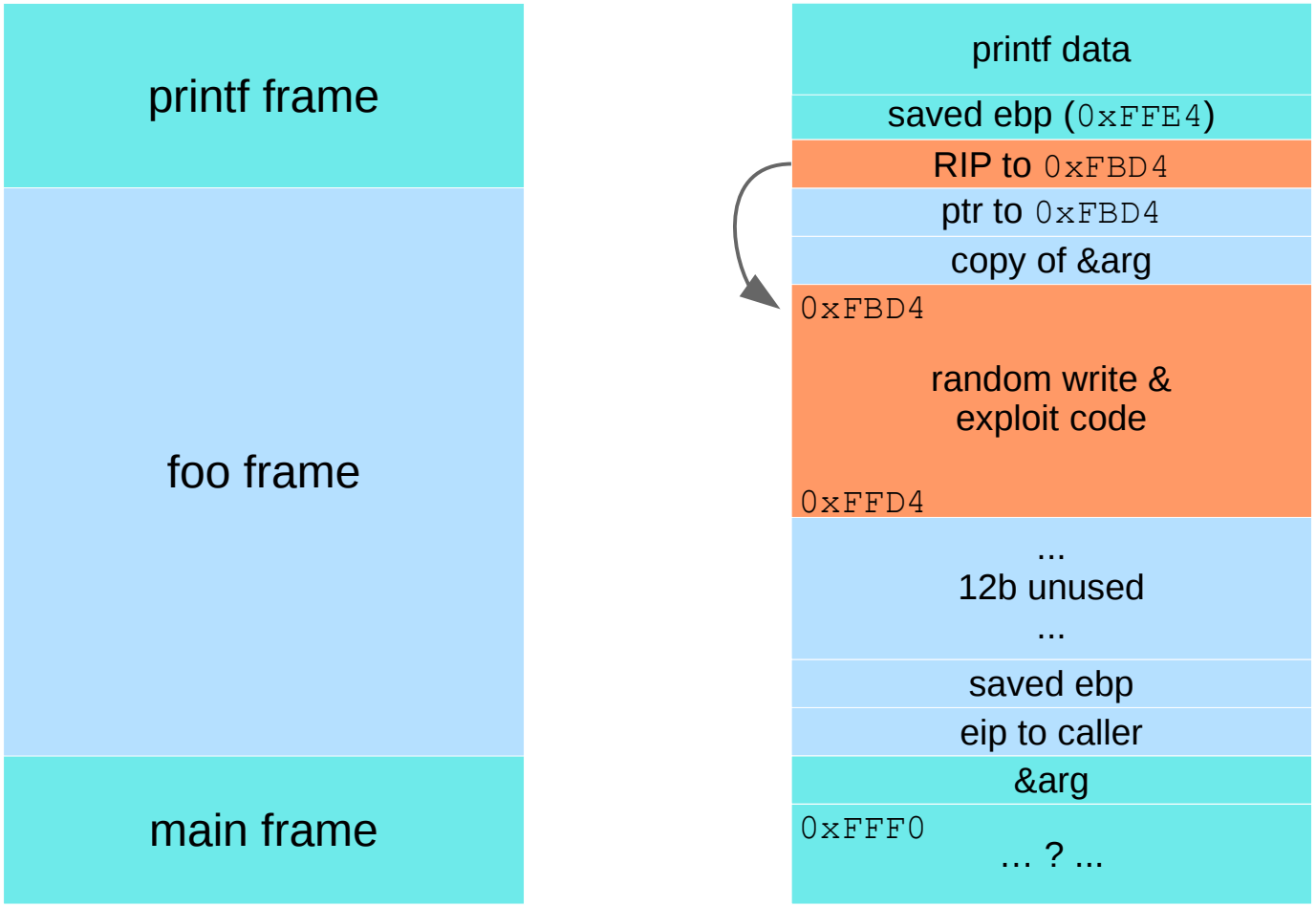



Running example

```
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

…
```

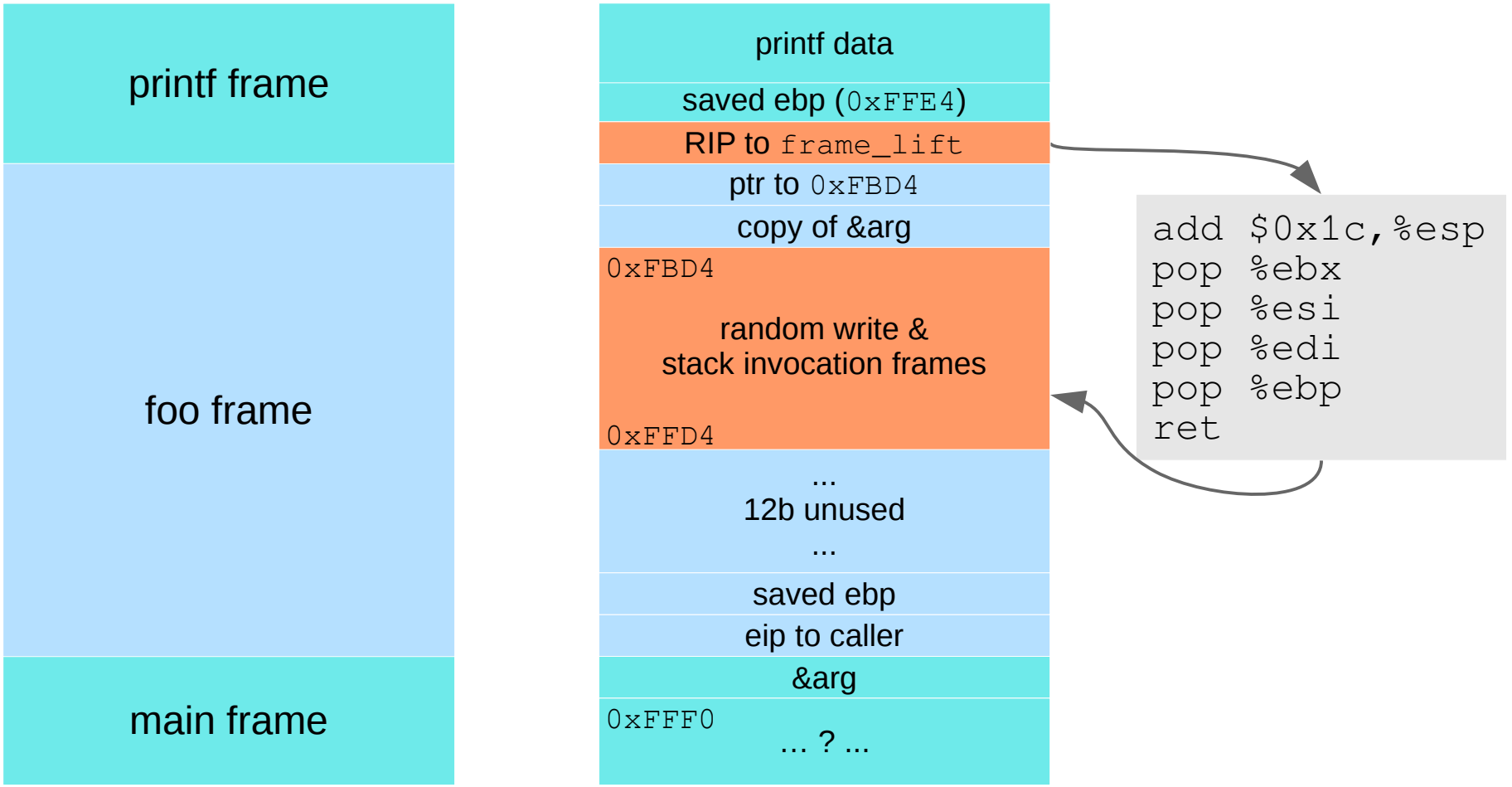
SOP: No Protection

- All addresses are known, no execution protection, no stack protection
  - Redirects control flow to code in the format string



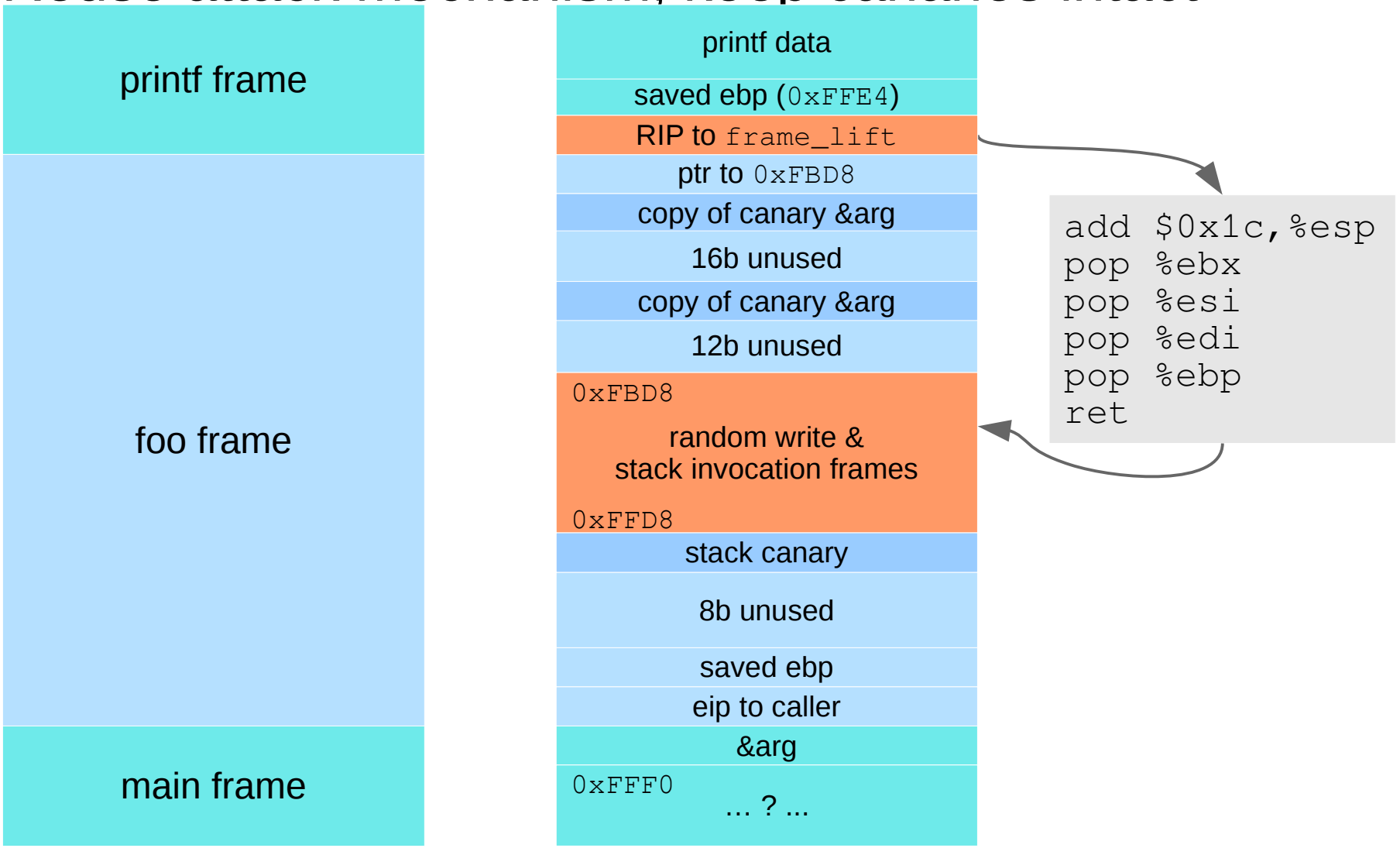
SOP: Only DEP

- 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

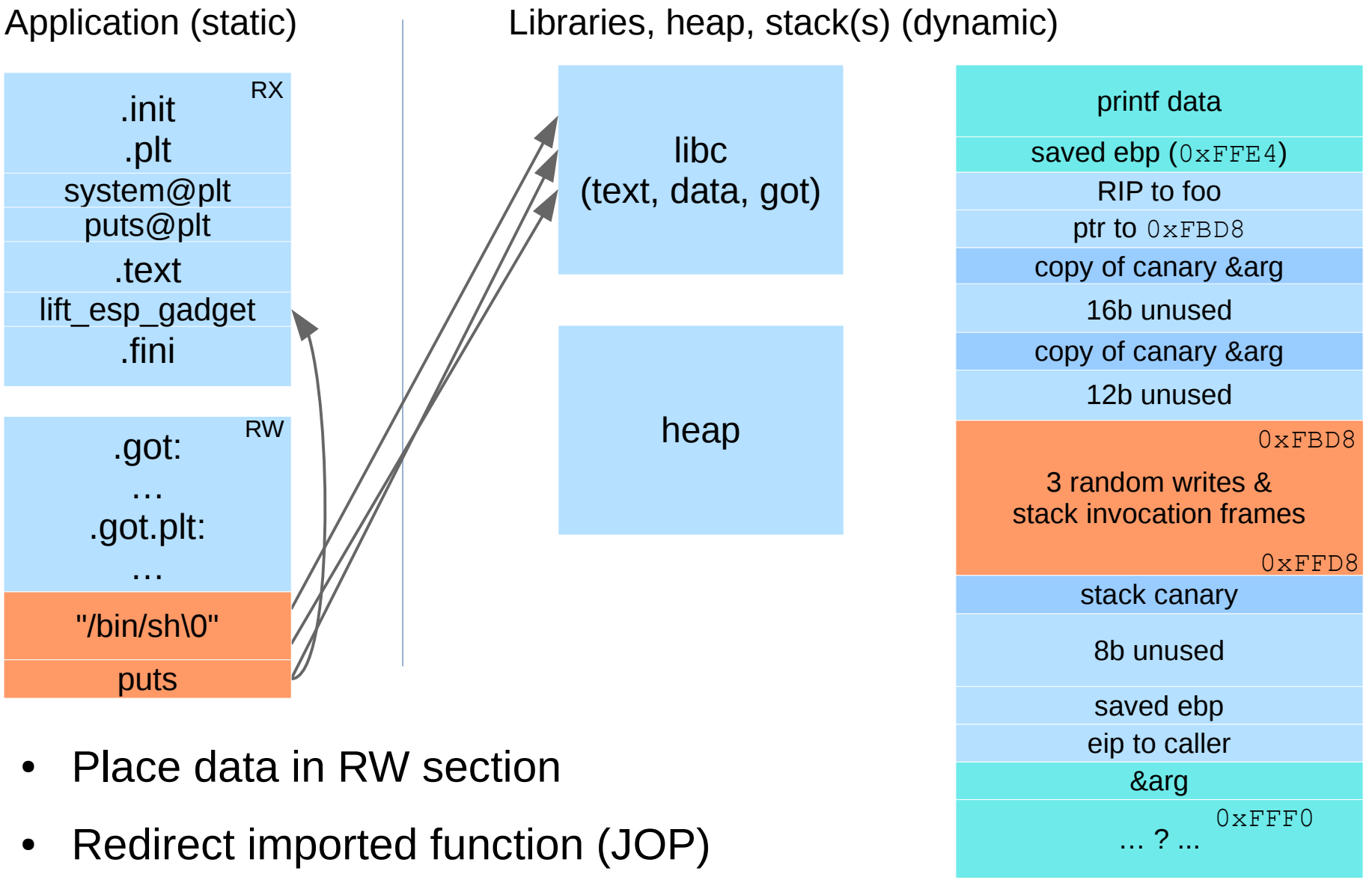


SOP: ASLR, DEP, ProPolice

- Combined defenses force SOP to reuse existing code
  - Static code sequences in the application object
  - Imported functions in the application (`.plt` and `.got`)

```
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