String Oriented Programming
Circumventing ASLR, DEP, and other Guards

Mathias Payer, ETH Zürich
## Motivation

Additional protection mechanisms prevent many existing attack vectors

<table>
<thead>
<tr>
<th>Feature</th>
<th>8.04 LTS (Hardy)</th>
<th>10.04 LTS (Lenny)</th>
<th>10.10 (Maverick)</th>
<th>11.04 (Natty)</th>
<th>11.10 (Oneiric)</th>
<th>11.04 (Precise)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Open Ports</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
</tr>
<tr>
<td>Password hashing</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
</tr>
<tr>
<td>SYN cookies</td>
<td>active</td>
<td>active</td>
<td>active</td>
<td>active</td>
<td>active</td>
<td>active</td>
</tr>
<tr>
<td>Filesystem</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Capabilities</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Configurable Firewall</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>PR_SET_SECCOMP</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>AppArmor</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>SELinux</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>SMACK</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Encrypted LVM</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>eCryptfs</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Stack Protector</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
</tr>
<tr>
<td>Heap Protector</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
</tr>
<tr>
<td>Pointer OBfuscation</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
<td>entry</td>
</tr>
<tr>
<td>Stack ASLR</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Lib/h/mmap ASLR</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Exec ASLR</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>bpf ASLR</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>VDSO ASLR</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
<td>kernel</td>
</tr>
<tr>
<td>Built as PIE</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Built with X86-64</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Built with SELinux</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Non-Executable Memory</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>/proc/sys/fs/file locking</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Link restrictions</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Hardcode scope</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Address protection</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Filesystem protection</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>/dev/kmem disabled</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Block module</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>loading data</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Stack protector</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Module RO/XN</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Kernel Address Display</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Blacklist file</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Networks</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
<tr>
<td>Syscall Filtering</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
<td>package list</td>
</tr>
</tbody>
</table>

-kernel: kernel is mandatory
-: optional
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*

Injests additional code into the runtime image

- Buffer overflow used to inject code as data

```c
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
  ```c
  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 $W \oplus X$
- 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 ($W \oplus X$) 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 $W \oplus X$
- 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 \( W \oplus X \)
- 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:

```c
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
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

```
--- main frame ---
0xFFF0 eip to caller
saved ebp (0xFFF)

--- foo frame ---
0xFFF0 RIP to foo
saved ebp (0xFFF)

--- printf frame ---
0xFFF0 printf data
```

```
--- exploit code ---
0xBFD4 random write & exploit code

--- stack ---
0xFFF0 &arg
0xFFF0 saved ebp
0xFFF0 eip to caller
0xFFF0 ?
```
SOP: Only DEP

DEP prevents code injection, rely on ROP/JOP instead

GNU C compiler adds frame_lift gadget

![Diagram showing the impact of frame_lift gadget on stack frames and printf data.]
ProPolice uses/enforces stack canaries

- Reuse attack mechanism, keep canaries intact

```
printf frame

foo frame

main frame

printf data
- saved ebp (0xFFF4)
- RIP to frame_lift
- ptr to 0xFBD8
- copy of canary &arg
- 16b unused
- copy of canary &arg
- 12b unused

0xFBD8
- random write & stack invocation frames

0xFFF8
- stack canary
- 8b unused
- saved ebp
- eip to caller
- &arg

0xFFF0
- ... ? ...
```

```
add $0x1c, %esp
pop %ebx
pop %esi
pop %edi
pop %ebp
ret
```
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

```c
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

Application (static)

- .init
- .plt
- system@plt
- puts@plt
- .text
- lift_esp_gadget
- .fini

- .got:
  - ...
  - .got.plt:
  - ... 

- "/bin/sh\0"
- puts

Libraries, heap, stack(s) (dynamic)

- libc
  - (text, data, got)

- heap

- printf frame

- foo frame

- main frame

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)