Lecture 4 : Address Space Layout Randomization (ASLR)

Alexandre Bartel

2019

Previously...

Previously... in Lecture 1 (Introduction)

- Software Development Life-cycle
- Vulnerability Life-cycle
- Vulnerability Disclosure

Previously... in Lecture 2 (Buffer Overflow)

- A buffer on the stack
- Return address on the stack
- Overwrite return address
- Jump to shellcode on the stack

Previously... in Lecture 3 (ROP)

- NX bit (stack non-executable)
- Gadgets in already loaded code
- Chain gadgets (addresses of gadgets and data on the stack)
- Only data on the stack

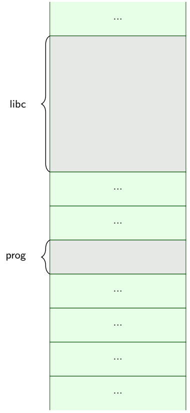
Why ASLR?

What is Address Space Layout Randomization?

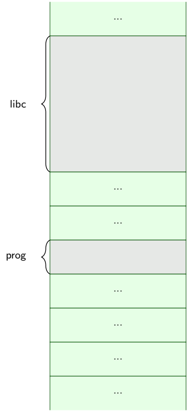
- A mechanism to load code segments at random addresses in memory.

What is Address Space Layout Randomization? (cont)

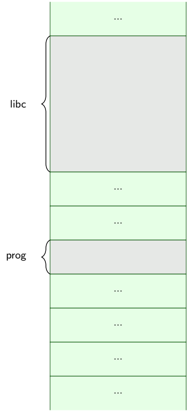
- Without ASLR, every code segment is loaded at the same address at every run of the program.



Program run 1



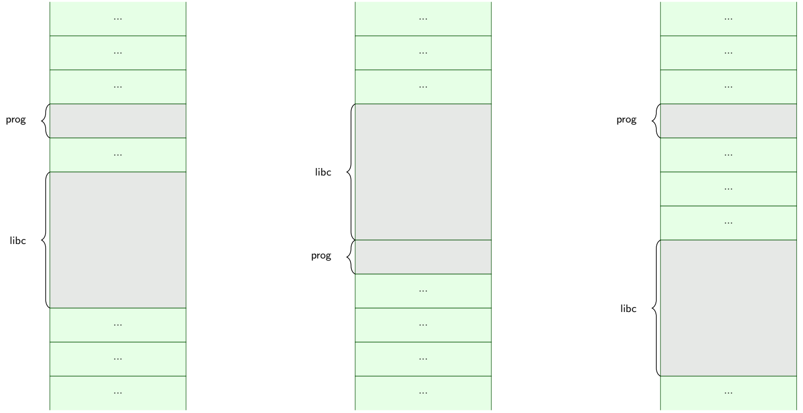
Program run 2



Program run 3

What is Address Space Layout Randomization? (cont)

- With ASLR, every code segment is loaded at a different random address at every run of the program.

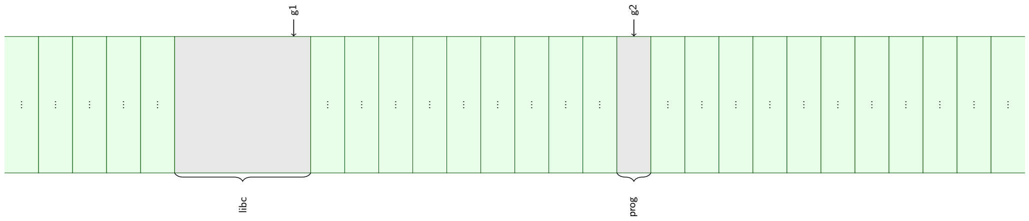


Program run 1 Program run 2 Program run 3

Why Address Space Layout Randomization?

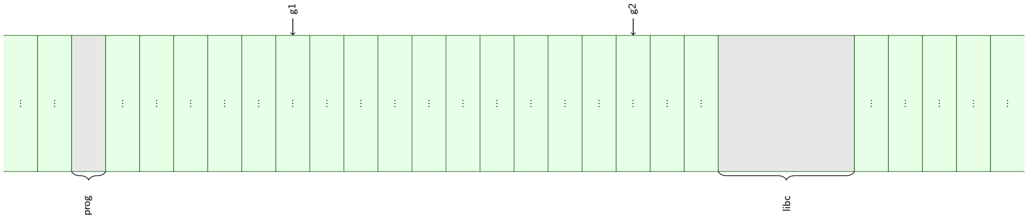
- It makes is (almost?) impossible to build and execute ROP chains.
- Indeed, addresses of gadgets change at every execution.
- With a wrong gadget address there is a very high probability that the program will crash...
- Launching one attack with n gadgets in m code segments on a target program running on a 64-bit system has a theoretical success probability of $\approx 1/(2^{64})^m$

Why Address Space Layout Randomization?



Without ASLR, addresses of gadgets $g1$ and $g2$ point to the instructions of $g1$ and $g2$ in code segments of the libc and the target program, respectively.

Why Address Space Layout Randomization?



With ASLR, addresses of gadgets cannot be pre-computed. Addresses of gadgets $g1$ and $g2$, computed previously, have a very high probability of NOT pointing to gadgets $g1$ and $g2$ anymore.

Problem Solved?

Problem Solved?

- ASLR works under the assumption that the attacker cannot know the addresses of the code segments
- Unfortunately, an attacker could know this information through an **information leak**

Information Leak

Definition

Information Leakage is a weakness where a software reveals sensitive data, such as technical details of the software, environment, or user-specific data. [1]

[1] Inspired from the definition available on Webappsec

Information Leak (cont)

Example: CVE-2017-14443

"An exploitable information leak vulnerability exists in Insteon Hub running firmware version 1012. The HTTP server implementation incorrectly checks the number of GET parameters supplied, leading to an arbitrarily controlled information leak on the whole device memory. An attacker can send an authenticated HTTP request to trigger this vulnerability." [1]

- Can an attacker bypass ASLR?
- What else is required for arbitrary code execution?

[1] https://nvd.nist.gov/vuln/detail/CVE-2017-14443

Information Leak (cont)

Example: CVE-2017-14443

"An exploitable information leak vulnerability exists in Insteon Hub running firmware version 1012. The HTTP server implementation incorrectly checks the number of GET parameters supplied, leading to an arbitrarily controlled information leak on the whole device memory. An attacker can send an authenticated HTTP request to trigger this vulnerability." [1]

- Can an attacker bypass ASLR?
- What else is required for arbitrary code execution?

[1] https://nvd.nist.gov/vuln/detail/CVE-2017-14443

Information Leak (cont)

Source Code Example (from Mitre)

```c
int getValueFromArray(int *array, int len, int index) {

  int value;
  // check that the array index is less than the maximum
  // length of the array
  if (index < len) {

    // get the value at the specified index of the array
    value = array[index];

  // if array index is invalid then output error message
  // and return value indicating error
  } else {
    printf("Value is: %d\n", array[index]);
    value = -1;
  }

  return value;
}
```

If the attacker controls *index* he/she can read outside the bounds of the array. This only works if the value returned by the function reaches the attacker somehow. For example, if the target process is a web server, the value could be used in a generated html page sent by the web-browser to the attacker

Information Leak (cont)

Example: CVE-2017-5753 (Spectre)

"Systems with microprocessors utilizing speculative execution and branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis. ." [1]

Example: CVE-2017-5754 (Meltdown)

"Systems with microprocessors utilizing speculative execution and indirect branch prediction may allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis of the data cache." [2]

[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5753
[2] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5754

Information Leak (cont)

Conclusion

With an information leak, an attacker could partially or completely analyze the memory of a process. This might include dumping all the code segments of the process. An information leak could thus break ASLR. An attacker can reuse the knowledge of the leak to exploit another vulnerability such as a buffer overflow via a ROP chain.

Information Leak (cont)

Limitations

Combining an information leak vulnerability with a software vulnerability mainly works if:

- The target software provides a scripting environment (ex: javascript in web-browsers).
- The target software can be "probed" multiple times (ex: web servers).

Question?

Lecture March 25th cancelled. (video instead)

Labs.