Hacking Blind | Blind ROP

Code Reuse Attacks and Defenses Against Them

Nicolas Frinker
History of Buffer Overflow Attacks

- Code injection
  - NX memory
  - ROP
  - ASLR

Brute Force ASLR (32bit)

+ Use additional vulnerabilities
  (information leakage)
ROP

- Local attack
- Binary available and can be analysed
- Build instruction sequence on stack to spawn a shell
ROP, Limitations and Defenses

- Rerandomization
- ASLR
- Stack Canaries
Remote ROP?

What if...

- ...the binary would not be available for analysis?
- ...the vulnerable service is running remotely on a server?
Blind ROP!

Requirements
● Stack vulnerability and knowledge, how to trigger it
● Identical binary running on multiple servers
  OR
  Service is automatically restarted on exit (e.g. by systemd)

Advantages
● Works on proprietary closed source binaries (without fuzz testing/reverse engineering)
● Works without knowledge of the source code or compilation details
● Generic exploit (not only for one specific binary)
Blind ROP & ROP defenses

- Rerandomization (only at reboot of system)
- ASLR (only for certain parts)
- NX memory (ROP => no code injection)
- Stack Canaries (BROP)

Scenarios
- Proprietary closed source software
- Open source libraries embedded in proprietary software (e.g. SSL)
- Manually compiled open source software
BROP, Attack Outline

- Find vulnerable remote service with buffer overflow
- Hope, the service is restarted or has multiple instances
- **Read the stack** bit by bit
- Find gadgets (**Blind ROP**) until write can be invoked
- Dump binary and build the exploit
Stack Reading

- Method already used for canary reading (exact values, 8 Byte)
- Also use to read Frame pointer + return address
- Not 100% reliable but we don’t need exact values anyway

- Defeated: Canaries
- Defeated: ASLR
- Found: Some valid address in the binary
Finding Gadgets

- Goal: Dump binary to our socket

- How to do this: Invoke syscall write with suitable parameters
  - Rdi: Socket
  - Rsi: buffer
  - Rdx: length
  - Rax: write syscall number

- Find ways to set these from stack (find suitable pop instructions followed by ret)
Finding Gadgets, Improved

- BROP gadget
- Procedure Linking Table (PLT)
  - write
  - strcmp

Makes the attack practically usable
The BROP gadget

- Restore callee saved registers (commonly found)
- Provides two gadgets
  - Pop rdi; ret
  - Pop rsi; ret
Procedure Linking Table (PLT)

- At the beginning of the binary
- Used to convert position independent function calls to absolute ones

- Scan for it from start (non-PIE, 0x400000) or backwards
- Skip bytes for performance

- Each entry x bytes, skip bytes during search for performance
- Most of them not crashing on false parameters (EFAULT)

- Find entries for strcmp and write
Identifying Gadgets

- Observe behaviour of connection
- Distinguishable outcomes:
  - Connection stays open, works normal
  - Connection stays open, but no reaction within timeout
  - Connection is closed by server

- Get distinguishable outcomes by preparing the stack
- Use stop and trap gadgets to get well known reaction
The Attack

- Use buffer overflow to read stack byte by byte
- Find suitable stop gadget
- Find BROP gadget
- Find PLT
- Identify strcmp and write calls in PLT
- Chain gadgets to dump binary to socket via write call
- The write system call

Additional challenges
- Guess file descriptor of socket (1-1024)
Implementation

- Meta-Exploit implemented in Braille

- Only requires function to automate attack:
  - \texttt{try\_exp(data)} -> CRASH, NO\_CRASH, INF
Performance & Stability

- About 4000 requests for a shell (< 20 minutes)
- Exploits do not rely on certain binary

- Successful attack against:
  - YaSSL library in mysql (20 min)
  - Nginx (1 min)
  - Some proprietary sample program
Limitations & Defenses

- Rerandomization after each crash (no fork, but execve)
- Small length of buffer overflow
- 48bit virtual addresses with zeros (64bit)
- Sleep on crash (Linux, NetBSD)
- Attacker needs to control last byte of overflow (e.g. no appended zero)
- Each connection needs to go to the same machine (e.g. no load balancer)
- Number of workers that got stuck (choose stop gadget wisely)
Multiple-Stacks countermeasure

- Proposed by team from Belgium
- Sort types of stack data into different categories
- Have a separate stack for every category

Advantages
- Near to no performance impact
- Does not rely on obscurity
- Defends against most ROP attacks
Conclusion

- Possible with BROP: Remote general purpose exploit defeating ASRL, Canaries, NX and also works on 64bit
- You could also use it locally
- Fully use PIE/ASLR
- Security by obscurity does not work well
- ASLR is only effective if used on full binary and rerandomization
References


Thank you.

- Is an automated firewall based protection possible?
- Do multiple stacks prevent attacks? Advantages? Drawbacks?