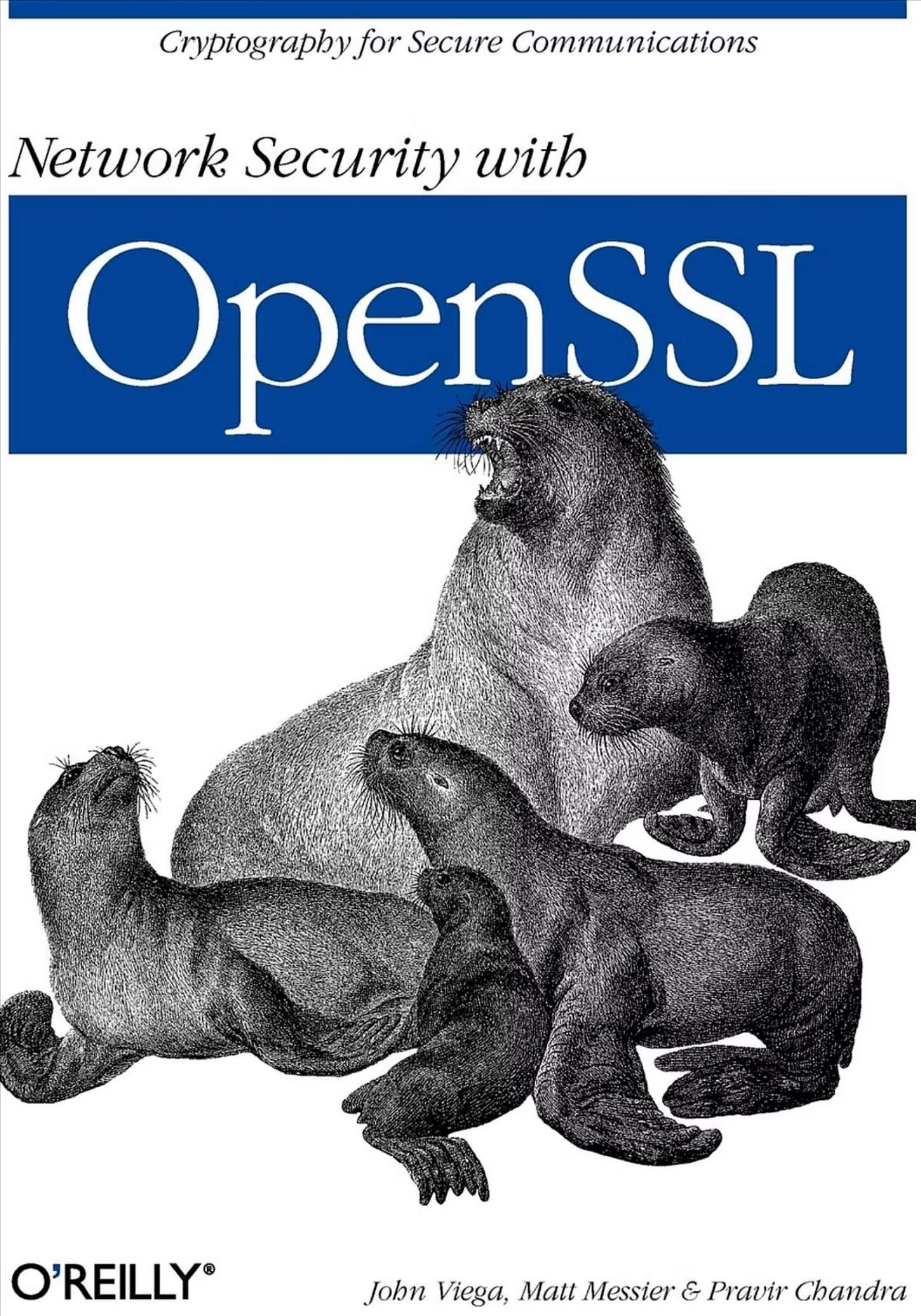
The Heartbleed Bug

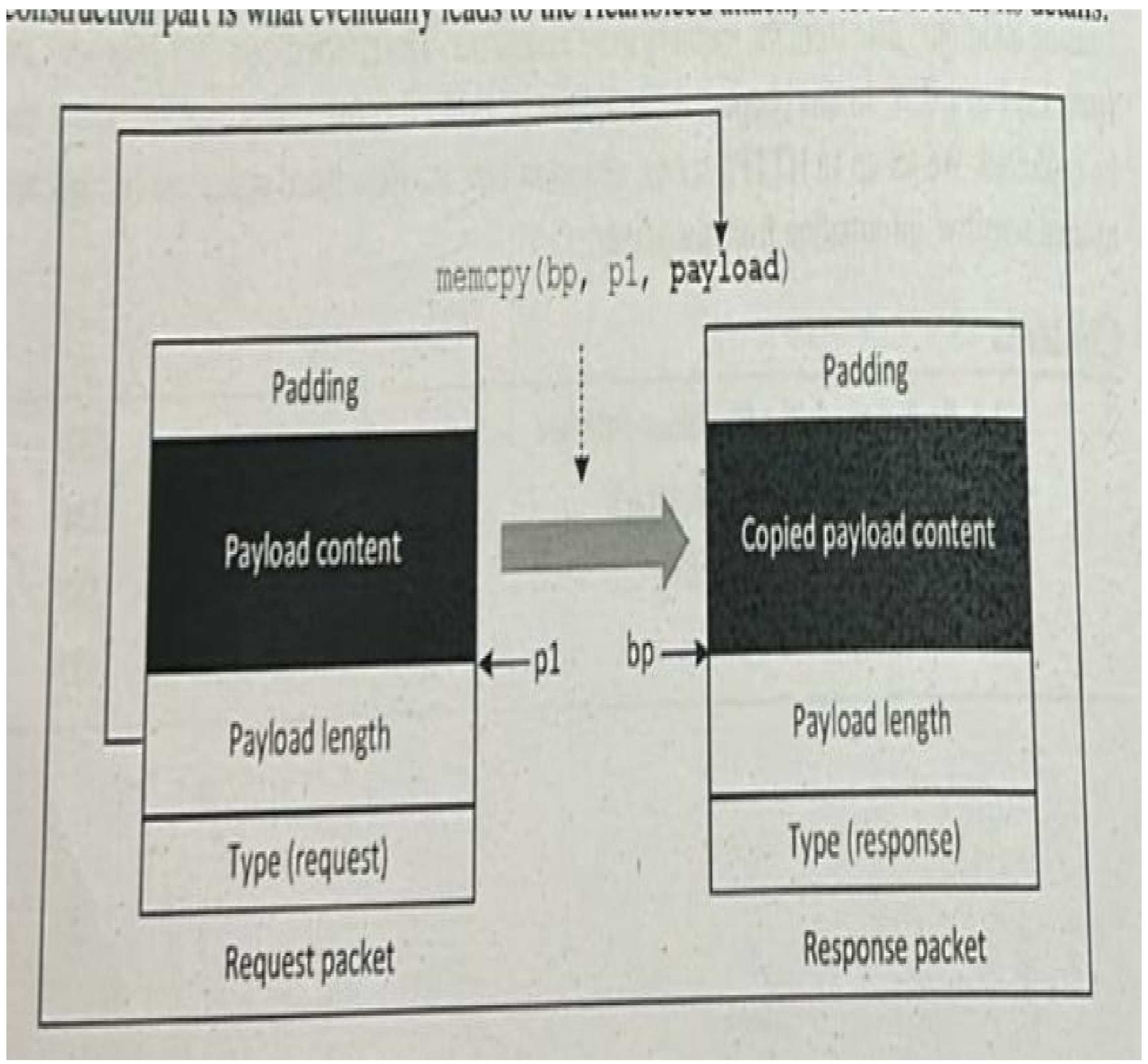
A critical security vulnerability that exposed sensitive data across the internet.


Cryptography for Secure Communications
Network Security with
OpenSSL
O'REILLY®
John Viega, Matt Messier & Pravir Chandra

What Is Heartbleed?

The Heartbleed bug (CVE-2014-0160) was a severe security vulnerability in OpenSSL, a widely used open-source library providing encrypted HTTPS connections for much of the internet. It allowed attackers to "bleed" sensitive data from a server's active memory, including usernames, passwords, session cookies, and critically, the server's private SSL key—the digital proof of identity that encrypts all traffic.

The flaw was not in the SSL/TLS protocol design itself, but in a specific implementation within OpenSSL's code.



The Heartbeat Protocol

To understand Heartbleed, you must first understand the legitimate feature it exploited: the Heartbeat Protocol. Creating a secure SSL/TLS connection is computationally expensive. To avoid dropping idle connections and paying that cost repeatedly, the Heartbeat protocol was created as a simple "keep-alive" check.

01

Client Sends Request

Client sends a Heartbeat Request packet containing a small payload and a field declaring the payload's length.

02

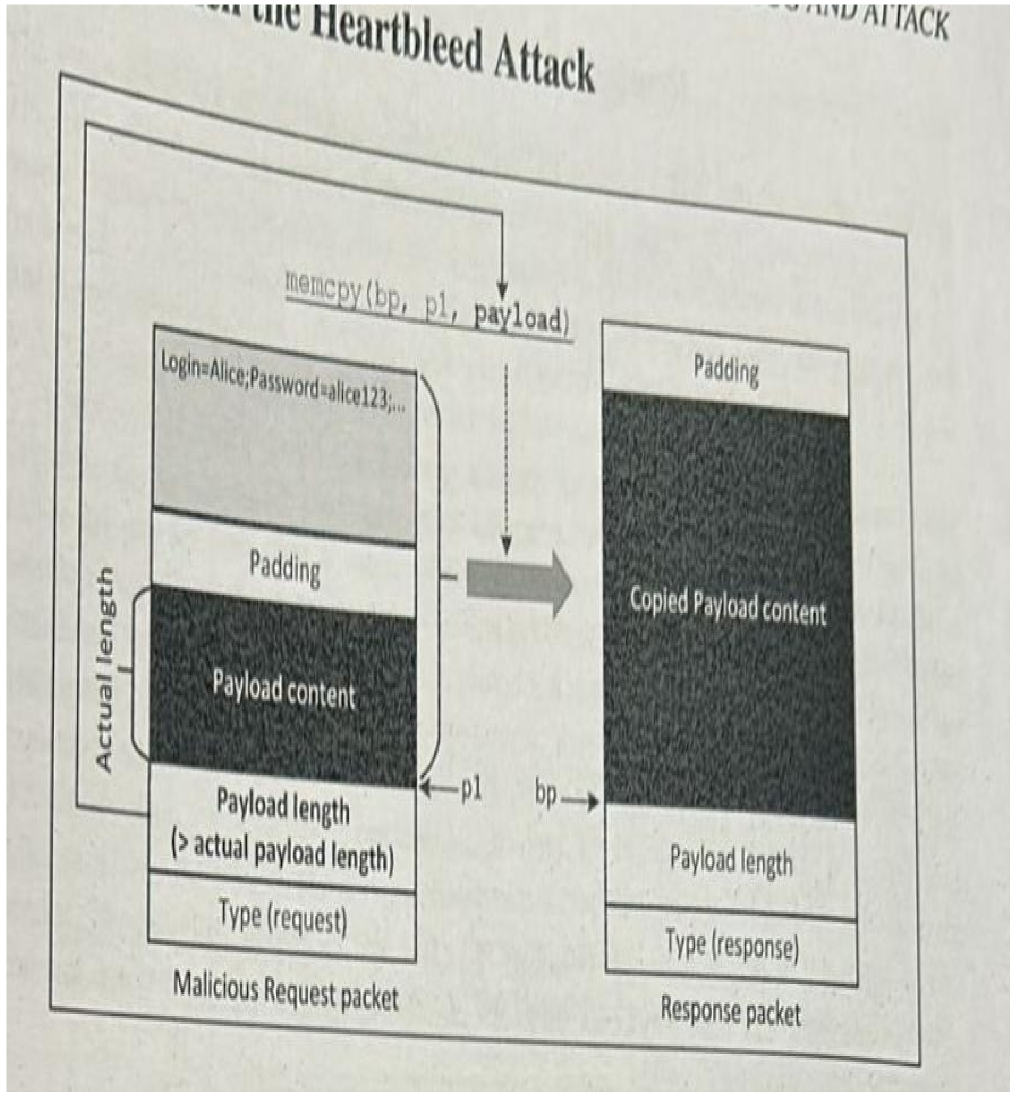
Server Responds

Server receives the request and sends back the exact same payload to confirm both sides are online.

03

Connection Maintained

This exchange keeps the connection alive without the overhead of creating a new secure connection.



The Fatal Flaw

The vulnerability was dangerously simple: OpenSSL's Heartbeat code was built on a fatal assumption—it blindly trusted the "Payload Length" field sent by the client. The server never verified whether the actual payload data matched the declared length.

This created a "buffer over-read" vulnerability. When the declared length exceeded the actual data, the server would copy whatever happened to be next in its memory, leaking sensitive information to the attacker.

How the Attack Works

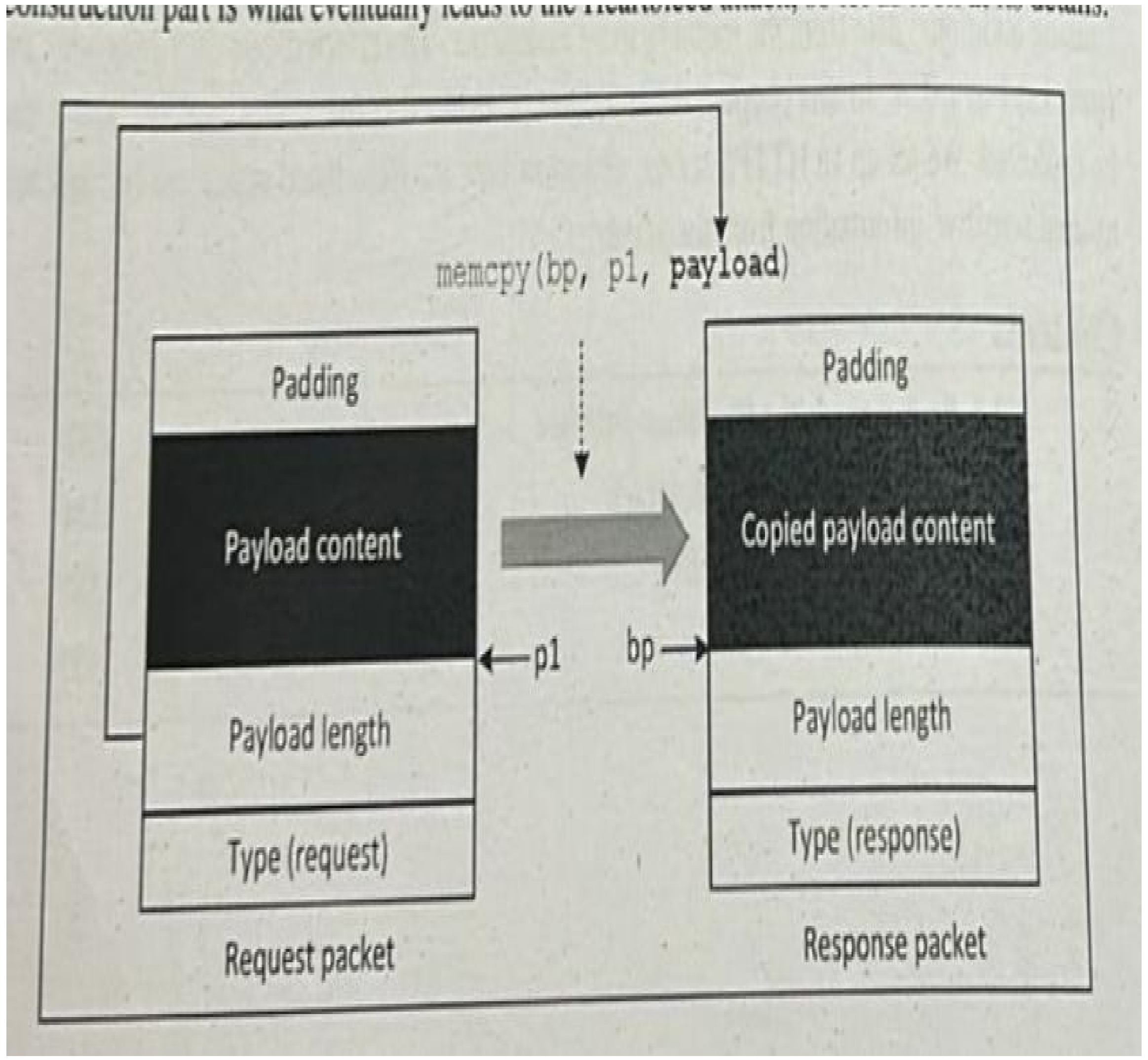
Normal Request

Declared length: 22 bytes. Actual payload: 22 bytes. Server responds normally. No data leaked.

Malicious Request

Declared length: 16,384 bytes. Actual payload: 22 bytes. Server copies 16,384 bytes from memory and sends it back.

The attacker sends the same small payload but lies about its size using a flag like -l 0x4000. The vulnerable server trusts this lie, allocates memory based on the false length, and copies far more data than actually exists in the request, bleeding server memory to the attacker.

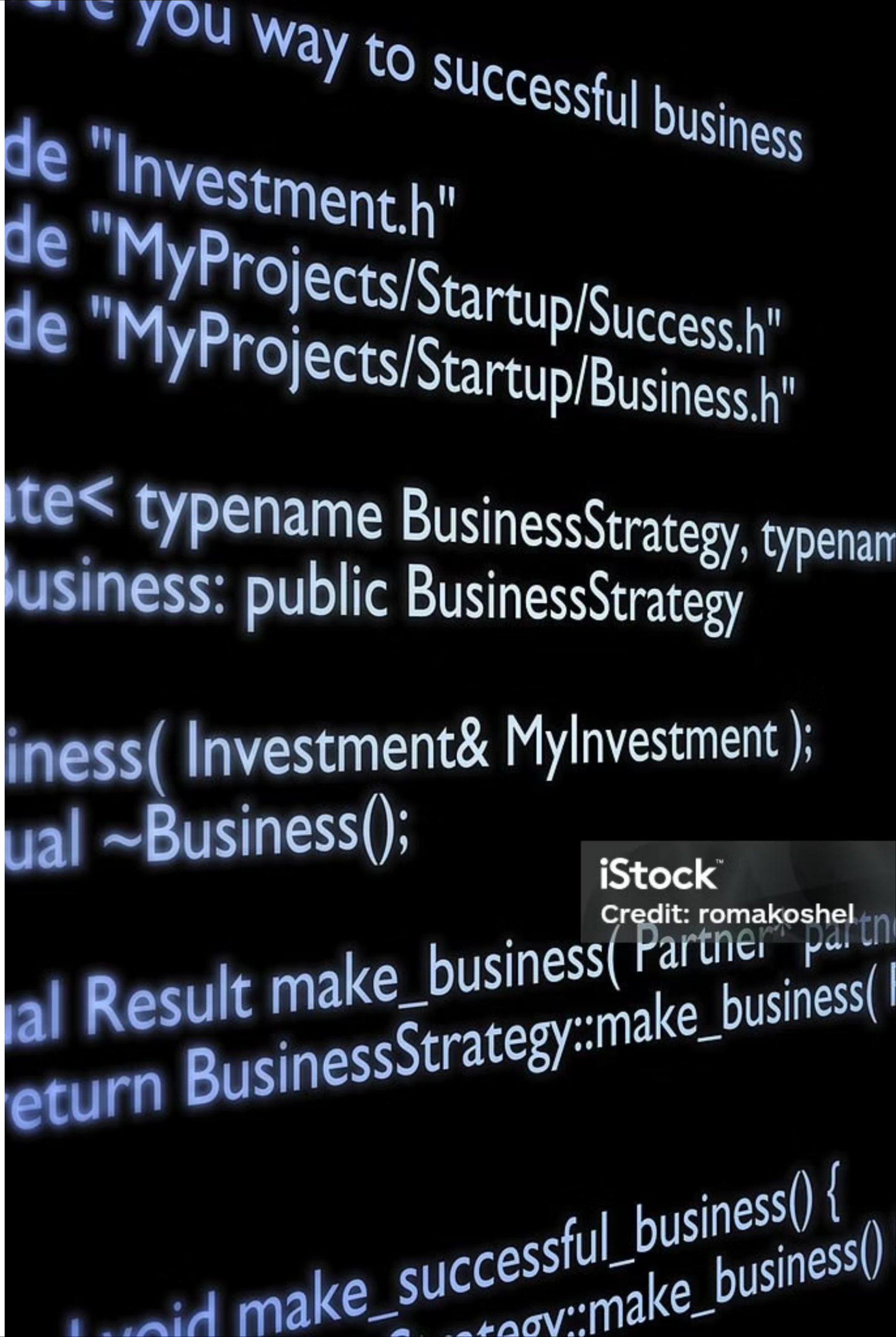


The Vulnerable Code

The flawed OpenSSL code demonstrates the problem:

```
n2s(p, payload);          /* Read untrusted length */
pl = p;                   /* Point to actual data */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
memcpy(bp, pl, payload);    /* FATAL: Copy payload bytes */
```

The code reads the untrusted payload length, then blindly copies that many bytes from memory without verifying the data actually exists. If the declared length exceeds the actual data, it copies whatever is next in the server's RAM—passwords, session tokens, private keys, and more.



What Was Stolen

When attackers exploited Heartbleed, the server's memory leak revealed devastating information:

HTTP Headers

Accept-Encoding, Referer, and other request metadata.

Session Data

Cookies and session tokens allowing account hijacking.

Credentials

Usernames and passwords from recent logins.

Private Keys

The server's SSL private key—the "crown jewel" proving identity and encrypting all traffic.

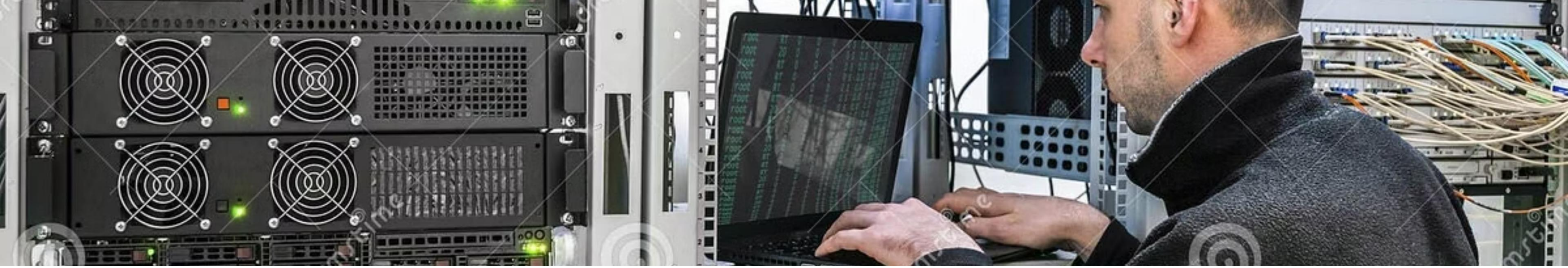
The Fix: Input Validation

The fix was elegantly simple—add the one thing that was missing: input validation. Before processing the Heartbeat request, OpenSSL developers added a critical check:

```
if (s->s3->rrec.length < 1 + 2 + payload) {    return 0;  /*
silently discard */}
```

This code asks: "Is the declared payload length larger than the entire packet I actually received?" If yes, it's a malicious lie, and the packet is discarded. If no, processing continues normally. This simple validation eliminated the vulnerability entirely.

Update baselines
Monitor for Vulnerability / release
ment
Patch management
rify
Deploy
Test & certify
© 2020 Destination Certification



Deployment and Response

For developers, the fix required updating OpenSSL to a patched version. For system administrators, the solution was straightforward:

1. **Update OpenSSL**

   Run sudo apt-get update to fetch the latest package information.

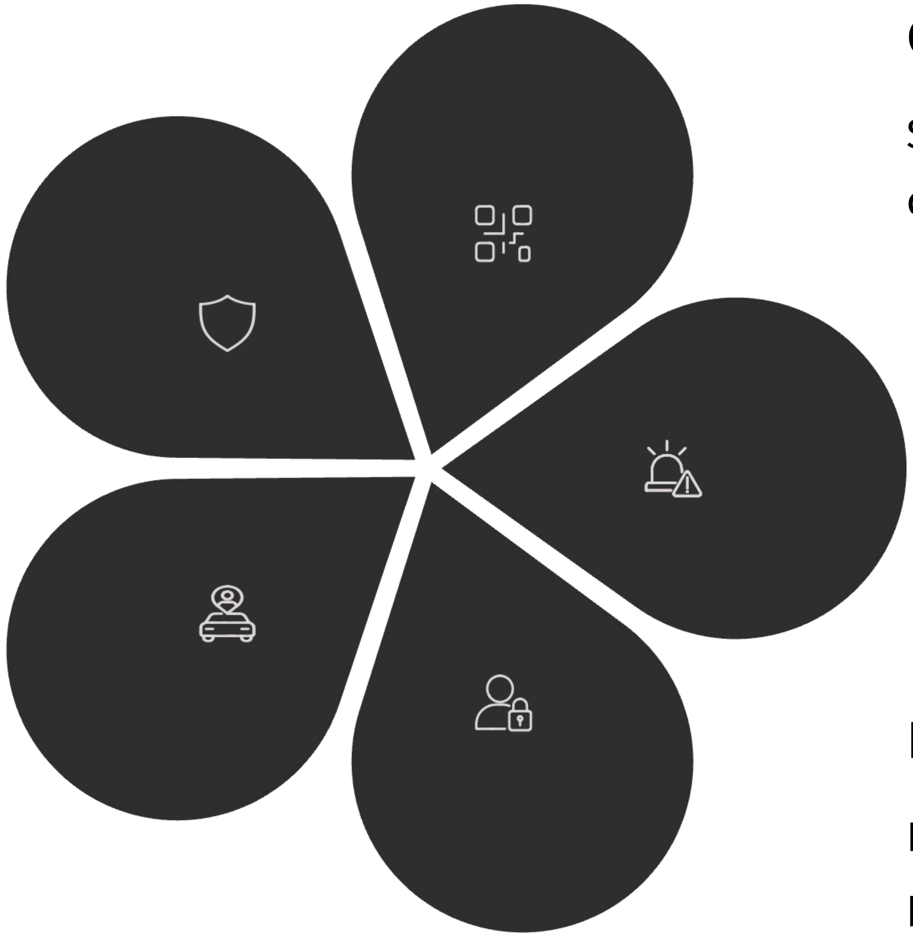
2. **Upgrade the Library**

   Run sudo apt-get upgrade to install the patched OpenSSL version.

3. **Regenerate Keys**

   Generate new SSL certificates and private keys to replace potentially compromised ones.

Key Lessons from Heartbleed



Never Trust Input

Always validate data from external sources, even in trusted protocols.

Transparency

Open-source security benefits from community scrutiny and collaborative fixes.

Code Review Matters

Simple implementation errors can have catastrophic security consequences.

Rapid Response

Quick patching and deployment are critical when vulnerabilities are discovered.

Proactive Security

Regenerate credentials after breaches to prevent ongoing exploitation.