Heartbleed - A wake-up call

1 Introduction

Last week the entire web discovered the existence of the so called "Heartbleed" vulnerability affecting one of most popular mechanisms used to secure communication with web sites. Starting with the first open announcements on 07th of April 2014, news about this security hole triggered the interest and concern of more or less everybody online. The publication created panic on the Internet due to its simplicity and efficiency. An attacker could remotely retrieve information from the memory of computers using the Transport Layer Security (TLS) features of OpenSSL, that is designed to provide authentication and encryption services for securing communication with web sites. This means that passwords, session cookies used in online authentication and private keys were potentially exposed for everyone to see, without the need for authentication.

As expected attackers started to try to find their way into vulnerable systems as soon as proof of concept tools exploiting the vulnerability were made available. Moreover as more information started to emerge, more impressively, it became clear that this vulnerability got unnoticed for almost 2 years putting under discussion the entire security life chain.

It also turned out that numerous, both commercial and non-commercial products, services and applications that utilise the faulty code could remain exposed to this vulnerability for an unknown period of time. The popularity of this specific implementation has led to the situation where pieces of this faulty code, even if labelled "open source", may be implemented in closed projects, products, or applications, which could be unfixed for a longer period of time.

Even though some major advisories and patches were quickly made available. Announcement to users and customers by affected services or suppliers were not in all cases handled sufficiently and transparent. Cisco, Blue Coat, Tor, VMware, and many others, sent out communication to their customers. Other services and companies have not issued an advisory or a patch until today.

2 The underlying mistake

(CAREFUL: some techie information ahead!)

The cause of the problem can be traced down to un-validated user input. Basically, the heartbeat feature of TLS is designed to keep a connection alive, one party in the communication provides arbitrary information and the other sends it back.

OpenSSL used a user-provided length field to allocate the memory buffer in which it copies the information provided by the initiating party. Unfortunately, there was no check if the 'length' field that was provided by the user was indeed the length of the data that was originally sent. When there was a mismatch between a small amount of data sent, but a big amount announced, OpenSSL then essentially returned information coming from the computer's memory. This information could include passwords, session cookies used in online authentication and private keys without the need for authentication.

As simple as it can sound, the fix that was implemented consisted simply in adding this consistency check before allocating memory.

3 Failure of the Quality Assurance process

The faulty code was submitted to the project by a volunteer programmer. As it is usual with open source projects, many programmers can submit code, but this code is then checked by maintainers who are responsible for Quality Assurance (QA). In this case, this process failed: the maintainer did not spot the bug, and the faulty code went live without notice.

The root cause is thus the combination of:

1. A programming mistake by lack of validation of input submitted by users.
2. In addition the use of insecure memory management routines, which is a problem caused by the development environment.
3. A failure in the QA process.

4 Other common programming mistakes

The Open Web Application Security Project (OWASP) ”Top programming mistakes” lists the most common programming mistakes. While it focuses mainly on web application programming, it still encompasses ”Unvalidated user input” even before other more famous mistakes such as ”faulty access control” and ”buffer overflows”. In 2014 we would expect that Quality Assurance and code review would avoid such simple vulnerabilities. Unfortunately major problems today include code injection, cross-site-scripting and –still – faulty authentication and session management.

5 Recommendations

5.1 Methodology

This problem could have been avoided by following programming guidelines that have been known for years, especially: all user input must be considered as untrusted until properly checked for consistency and validity. This is still nowadays a common programming mistake and points to a fundamental lack of security awareness on developer level. Our first recommendation is therefore to **use secure development methodologies and increase the awareness of developers about programming mistakes and how to avoid them**.

5.2 Coding standards

The C programming language is the most used programming language for systems-level development. It exists since decades, was developed with performance in mind, not security. It does not protect the programmers against mistakes or unforeseen circumstances. Security is not a feature of the libraries provided by the language and its development environment. It is too easy to introduce mistakes into the code in small projects, and even easier in huge projects with millions of lines of code (and tight deadlines). Our second recommendation is **to follow and enforce secure coding principles**, like those proposed by CERT/CC and others.

5.3 Quality assurance

Before source code is used in a product, or an application, it is strongly recommended to review and test the code, and if necessary to make security improvements, even if that would influence the publication date. This is also true for open source project, were a common motto is that “given enough eyeballs, all bugs are shallow”. The Heartbleed case shows that in practice, open source software, like any software, needs to go through well-defined quality assurance processes. Our third recommendation: **any software project must include security in the whole development process**.

5.4 Vulnerability communication

The communication process regarding the Heartbleed vulnerability has been, and still is, too fragmented. Vendor response is not synchronised and does not always provide actionable information. Also the level of detail of communicated information varies greatly from vendor to vendor, as is the audience addressed. Some vendors only provide information for service providers, others only for website owners, and information for the public at large is left to the mass media without guidance. Our fourth recommendation: **establish standards, guidelines and (where appropriate) coordination for practical, non-headline-seeking information sharing regarding serious vulnerabilities**.

6 About “Info Notes” from ENISA

With the “Info Notes” series ENISA aims at giving the interested reader some background and recommendations about NIS related topics. The background and recommendations are derived from past experiences and common sense, and should be taken as starting points for discussions on possible course of action by relevant stakeholders. Feel free to get in touch with ENISA to discuss or inquire more information on the “Info Notes” series (cert-relations@enisa.europa.eu).