Trustworthy Software: lessons from 'goto fail' & Heartbleed bugs

***H A Boyes**[*], **P Norris**[†], **I Bryant**[*], **T Watson**[*]*

[*] *Cyber Security Centre, WMG, University of Warwick, UK. (hb|ib|tw@warwick.ac.uk)*
[†] *Cyber Security Centre. De Montfort University, UK (pdn@dmu.ac.uk)*

Keywords: Heartbleed, trustworthy software, software defect, embedded systems, cyber security.

Abstract

In the first four months of 2014, two major vulnerabilities were announced affecting operation of the Transport Layer Security (TLS) protocol, which is used by applications to secure Internet communications. The 'goto fail' bug affected Apple's iOS and OS X software and the 'Heartbleed' bug affected versions of the OpenSSL software. Whilst the Apple bug was serious because it affected a wide range of Apple products, the Heartbleed bug was of greater significance due to widespread use of the OpenSSL library. This paper considers the lessons to be learned from these incidents. It examines how the use of the Trustworthy Software Framework (TSF) developed by the authors could have helped to reduce the risk of a major bugs like 'goto fail' and Heartbleed. It also examines the responsibilities of developers where they use third party libraries and the need for appropriate due diligence. The paper also makes recommendations about how incidents like this should be handled to avoid confusing and contradictory messages being given.

1 Introduction

Security of Internet communications, including message authentication and message integrity, are key features required to undertake many transactions. Without these features there are risks of fraud, impersonation and loss of sensitive commercial or personal information. The Transport Layer Security (TLS) protocol [1] provides cryptographic protection of Internet communications through the production and exchange of a session key, which is then used to encrypt data between the two parties. The protocol is in widespread use, protecting applications such as web browsing, email, instant messaging, VOIP, enabling virtual private networks (VPNs) and protecting access to embedded systems software.

In the first half of 2014, there were two serious cyber security incidents related to errors in the software implementation of the TLS protocol. The first, in February, related to Apple Inc's implementation of the protocol in its operating systems, this defect was referred to by the media as the 'goto fail' bug. The second, in April, concerned a defect in seven versions of the OpenSSL implementation, this defect was widely referred to in the media as the Heartbleed bug.

This paper examines both bugs and the failure of the developers to detect them before they were incorporated into live systems or applications. Responses from the developer community are considered, as is coverage of the bugs in the media. Both bugs raise serious questions about software quality, whether developed commercially as in Apple's case or as part of an open source project such as OpenSSL. The paper considers the lessons to be learned from both incidents. It examines how use of the Trustworthy Software Framework (TSF) could help developers to reduce the risk of a major bugs like 'goto fail' and Heartbleed. It also examines the responsibilities of developers where they use third party libraries, such as the OpenSSL library, and the need for appropriate due diligence. The paper makes some recommendations about how incidents like this should be handled to avoid confusing and contradictory messages being given.

2 The Transport Layer Security (TLS) protocol

The Transport Layer Security (TLS) protocol was first defined in 1999 [2] as Version 1.0, which was obsoleted by Version 1.1 in 2006 [3], which in turn was made obsolete by Version 1.2 in August 2008 [1]. The protocol provides communications security over the Internet by allowing client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery. The protocol is independent of the application(s) using it, and is intended to deliver both privacy and data integrity between the two communicating applications. The protocol comprises two layers: the TLS Record Protocol and the TLS Handshake Protocol.

The TLS Record Protocol operates at the lowest level, providing connection security that has two basic properties, the connection is private and reliable. It is used to encapsulate various higher-level protocols. One such protocol is the TLS Handshake Protocol, which allows the server and client to authenticate each other, and to negotiate an encryption algorithm and cryptographic keys before the application transmits or receives any data.

The TLS Handshake Protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated;
- The negotiation of a shared secret is secure, i.e. cannot be accessed by an eavesdropper;

- The negotiation is reliable, i.e. cannot be modified without detection.

The TLS protocol has a number of extensions [4], one of which is the Heartbeat extension [5] that delivers keep-alive functionality without performing any renegotiation. This allows a secure connection to be maintained for a period where there is not continuous data transfer. For example, when a user is completing a secure web form there may be no traffic between the server and the browser from the point where the form is displayed in the browser until the user submits the form. The Heartbeat protocol runs on top of the Record Protocol and consists of two message types: HeartbeatRequest and HeartbeatResponse. When one of the communicating parties issues a HeartbeatRequest message, the other party should immediately respond with a HeartbeatResponse message.

3 Apple's 'goto fail' bug

A serious bug had been present in the Apple's operating system software for over a year affecting the implementation of the TLS protocol, which prevented the checking of digital signature certificates, and thus prevented the authentication of a peer's identity. The Apple 'goto fail' bug became public knowledge on 21 February 2014, when Apple posted a security advisory [6, 7] and issued an urgent security update for the vulnerable iOS software. The affected software was running on a number of Apple platforms, including iPhones 4 and 5, iPod touch (5th generation), and iPad 2. The advisory stated the impact was that "An attacker with a privileged network position may capture or modify data in sessions protected by SSL/TLS". The cause of the problem was described as the secure transport protocol was failing to validate the authenticity of the connection. This was being addressed in the update by restoring the missing validation steps.

In addition to the impacted iOS software, the bug also affected some versions of Apple TV, and versions of the Apple OS X 10.9.x, which runs on Mac Server and MacBook product lines. The Mitre CVE filing [8] indicated that the cause of the vulnerability was that the software did "not check the signature in a TLS Server Key Exchange message, which allows man-in-the-middle attackers to spoof SSL servers by (1) using an arbitrary private key for the signing step or (2) omitting the signing step".

An analysis of the source of the defect published the following day [9] included a quote from Apple's published code [10], see Listing 1. The bug was caused by the presence of the repeated 'goto fail;' line. The first 'goto fail' is correctly associated with the 'if' statement but the second is not conditional, therefore causing the code to always jump to the 'fail' label from the second 'goto fail line'. At this point in the code, the SHA1 update operation will have been successful, and the value of the 'err' variable will therefore always contain a successful value so the signature verification can never fail. As a consequence of this error, there is no proof that the server possesses the private key matching the public key in its certificate.

```
static OSStatus
SSLVerifySignedServerKeyExchange(
    SSLContext *ctx, bool isRsa, SSLBuffer
     signedParams, uint8_t *signature,
    UInt16 signatureLen)
{
        OSStatus          err;
        ...

        if ((err = SSLHashSHA1.update(&
            hashCtx, &serverRandom)) != 0)
                goto fail;
        if ((err = SSLHashSHA1.update(&
            hashCtx, &signedParams)) != 0)
                goto fail;
                goto fail;
        if ((err = SSLHashSHA1.final(&
            hashCtx, &hashOut)) != 0)
                goto fail;
        ...
fail:
        SSLFreeBuffer(&signedHashes);
        SSLFreeBuffer(&hashCtx);
        return err;
}
```

Listing1 - Partial listing of sslKeyExchange.c

On 25 February, Apple released OS X 10.9.2 [11, 12] to patch the defect in this product range. The bug was widely covered, explaining that it could result in snooping on sensitive transactions through man-in-the-middle attacks, and consumers were advised to download and install the updates [13–15]. Until the patch was installed consumers were advised to avoid untrustworthy network and WiFi connections when undertaking sensitive transactions, e.g. online banking. Reports highlighted how it was "painfully easy" to mount such attacks on any public network, e.g. the attacker could be in "the same Starbucks as you" [11].

4 OpenSSL's Heartbleed bug

Barely six weeks after the announcement of the 'goto fail' bug, the Heartbleed bug was making news. The bug simultaneously discovered by Codenomicon and Neel Mehta of Google Security [16] was disclosed on 7 April. The bug even had its own logo, see Figure 1, and received significant coverage [17–20]. Compared to the 'goto fail' bug the headlines were more dramatic. Security researchers highlighted a number of vulnerable websites, including Twitter, GitHub, Yahoo, Tumblr, Steam, Flickr, HypoVereinsbank, PostFinance, Regents Bank, Commonwealth Bank of Australia, and the anonymous search engine DuckDuckGo [21]. It was reported that "around two-thirds of websites are vulnerable to so-called 'heartbleed hackers'", and facilities to check websites for Heartbleed vulnerability were reported in the mainstream media [22]. Conflicting

advice was offered to the public – either to change all their passwords [23], or to wait until an affected website has been fixed before logging in to change passwords [24–26]. There was some confusion over the nature of Heartbleed, with some news reports referring it as a virus [27]. The bug particularly caught UK media attention when Mumsnet was hacked [28].

Figure 1 - Heartbleed logo ©Codenomicon

The Heartbleed bug affected the TLS implementation in OpenSSL 1.0.1 before version 1.0.1g. These versions do not properly handle Heartbeat Extension packets, which therefore allows remote attackers to obtain sensitive information from process memory via crafted packets that trigger a buffer over-read [29]. The bug was caused by improper input validation as a result of a missing bounds check in the handling of the Transport Layer Security (TLS) Heartbeat extension. A "Heartbeat Request" message comprises a payload, e.g. a text string, and the payload's length is represented as a 16-bit integer. When the receiving computer receives a "Heartbeat Request" message it should return the exact payload to the sending computer. By creating a malformed payload where the payload length was greater than the length of the actual payload, due to the code's failure to perform a failure to bounds check, an attacker can read up to 64KB of memory. This potentially allows retrieval of sensitive or privileged information including encryption secret keys, usernames, passwords and user data.

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0; /* silently discard */
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.
    length)
    return 0; /* silently discard per RFC
        6520 sec. 4 */
pl = p;
```

Listing 2 - Code added to fix the Heartbleed bug

The problem occurred in two modules (ssl/d1_both.c and ssl/t1_lib.c) and was fixed by addition of six lines of code to both, see Listing 2. This code does two things. The first check stops zero-length heartbeats. The second check checks to make sure that the actual record length is sufficiently long [30]. Effectively the fix ensures that the software will ignore Heartbeat Request messages that deliver more data than provided by their payload.

Unfortunately the Heartbleed problem was not limited to websites as the OpenSSL code was embedded in a number of products, including IP phone systems, video conferencing equipment, VPN software and consumer routers [31–33]. Possibly the most serious systems vulnerabilities identified are in industrial control systems, with many major vendors impacted by the bug [34].

5 The Trustworthy Software Framework

Through the Trustworthy Software Initiative (TSI) [35] the authors developed the Trustworthy Software Framework (TSF) [36], a reference to the existing body of knowledge, which includes functional safety, information security, and systems and software engineering, which provides a consensus collation of good practice for software trustworthiness. The TSI has defined software trustworthiness in terms of five characteristics:

- Safety – the ability to operate without harmful states;
- Reliability – the ability to deliver functionality or services as specified;
- Availability – the ability to deliver functionality or services when required;
- Resilience – the ability to transform, renew and recover in timely response to events;
- Security – the ability to remain protected against accidental or deliberate attacks.

The TSF recognises that the level of trustworthiness required will vary according to the software's planned or actual use. It therefore requires the software to be considered in terms of the role it plays in the overall system or service, and the impact that any defect or deviation would have on the performance of that system or service. A formal assessment should be used to establish the required Trustworthiness Level (TL) of the software [36]. For example, where a software component has critical or significant impact on system functionality and its role is paramount, i.e. the software provides the sole source of trustworthiness in the component, sub-system or system, then this would require the highest level of software trustworthiness (TL4) to be delivered through predicable processes.

The TSF is based on four core concepts:

- Governance – the need to establish confidence in the trustworthiness of the software, which is achieved by having appropriate governance and management arrangements in place to address risk, control and compliance. The appropriateness of governance measures will be determined by

the stakeholders' needs and the environment in which the software is used.

- Risk – an assessment of the risk that the software will fail to meet the users' and stakeholders' needs. This includes scoping those risks that are influenced by external dependencies, understanding the consequences of any software failure, error or non-performance in view of the adversities (risks and hazards) that may be faced and ways in which the software may be susceptible.
- Controls – software trustworthiness is achieved through the application of risk management, to enable identification and, where practical, the elimination of risks through the use of appropriate controls. These controls typically fall into one of the following categories: personnel, physical, procedural and technical.
- Compliance – having adopted governance measures, understood the relevant risks and decided what control measures to adopt, developers and users of the software are required to implement the governance regime and apply the control measures.

The TSF has used these concepts to define a set of twenty nine principles related to software engineering good practice [36].

6 Discussion

In any application where it is implemented, the TLS software is a fundamental security component that provides security of communications and assurance of communications integrity. When assessing the required trustworthiness level of TLS software, it would be reasonable to assert that any failure of TLS functionality is likely to have a critical impact and that it has a paramount role in maintaining communications security. For example, when a customer purchases goods online with a credit card or uses online banking services, the TLS software provides the sole source of trust that exchanges between the user's web browser and the relevant web server are protected from interception and fraud. Given the high level of trustworthiness required of this software, users would expect the software developers to have delivered bug free code.

There has been public analysis and comment on the code quality and software development practices related to both bugs. Of the twenty nine TSF principles referred to above, three controls are of particular significance to both 'goto fail' and Heartbleed. These are choice of appropriate tools, the practice of hygienic coding and the performance of internal pre-release review.

6.1 Choice of appropriate tools

The software development tools used throughout the software lifecycle should be appropriate for the level of trustworthiness that needs to be achieved. For code that requires the highest level of trustworthiness, the choice of programming language and compilers, the use of integrated development environments (IDEs) and test tools, and appropriate configuration of all the tools used can contribute to improved code quality.

In the case of the Apple software, the code between the second 'goto fail' line and the 'fail' label statement was dead or unreachable code. This could have been detected when the code was compiled if the compiler had been set up to warn about unreachable code, i.e. the compiler needs to be correctly configured to flag this type of issue [9].

Alternatively, if during unit and system level testing developers had used appropriate test tools, e.g. a static analysis or coverage analysis tools, they could have identified a lack of test coverage of these unreachable lines. The use of an appropriate development environment, such as the Eclipse IDE, could also have helped with the formatting of the code, by automatically enforcing code formatting when the code is saved [37].

Following notification of the Heartbleed bug and the subsequent increased scrutiny of the OpenSSL code, the OpenBSD project team announced that they were going to fork and refactor the code [38]. There were clearly serious issues with the quality of the OpenSSL code, as within days the OpenBSD team reported that they had "already removed 90,000 lines of C code and 150,000 lines of content" [39]. This swift removal of a large volume of redundant code and other content raises serious questions about the governance regime and the quality of the tools used to manage the code base. Indeed the presence of such a large volume of extraneous code would hamper the use of a number of test tools due to the 'noise' created by errors or warnings from the unwanted code obscuring new errors in recently written or modified code.

6.2 Practice of hygienic coding

To reduce the degree to which defects may occur or be exploited, production of software should be in accordance with clear coding standards. Hygienic coding practice include formatting of code, initialisation of variables, validating inputs and outputs, error handling, naming conventions, explicit management of resource access and removal of detritus.

The code affected by the 'goto fail' bug was poorly formatted, which made it easier for errors to be missed during code reviews [9, 37]. As illustrated in Listing 3, if the coding style made proper use of white space, tabs, newlines and curly braces "{ }", it would have been much easier to spot the repeated line. Code formatting is an important security feature, which contributes to the readability and understanding of the code. If done by hand, formatting of code can be tedious and error prone, however with the right choice of development tools it can be automated.

Whilst in the C programming language it is not mandatory to use curly braces as part of an 'if' statement, applying appropriate coding standards reduces the opportunity for this type of bug. For example, if the coding standard require that "Braces shall always surround the blocks of code (a.k.a. compound statements), following if, else, switch, while, do

and for statements" then the presence of the second 'goto fail' line would be much easier to spot during any code review [40].

```
    if ((err = SSLHashSHA1.update(&
        hashCtx, &serverRandom)) != 0) {
        goto fail;
    }

    if ((err = SSLHashSHA1.update(&
        hashCtx, &signedParams)) != 0) {
        goto fail;
    }

    goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx
        , &hashOut)) != 0) {
        goto fail;
    }
```

Listing3 - Illustrating improved formatting of code

The OpenSSL software suffered from a number of hygienic coding issues, not least the volume of extraneous code mentioned in Section 6.1. The presence of this code represents a serious issue, which led to the accusation that OpenSSL as a project was "not developed by a responsible team" [41]. The OpenBSD team also identified with the way that the OpenSSL team had managed memory allocation, including the development of code which bypassed or replaced core functionality and therefore undermined the potential detection of the error.

The Heartbleed bug highlights a key security failure that underpins a number of security vulnerabilities (e.g. SQL injection and cross-site scripting), namely the need to validate inputs and outputs. The concepts of out-of-bounds reads and buffer over-reads are well known weaknesses in the C programming language [42, 43], which should therefore be explicitly addressed when reading from buffers and memory. As illustrated in Listing 2, only a few lines of code were actually necessary to prevent the bug.

6.3 Performance of internal pre-release review

To deliver trustworthy software the TSF recommends that organisations perform internal pre-release reviews. These reviews should form part of an integration and test process, which encompasses QA testing, load and performance testing, regression and acceptance testing prior to release for customer or public use. The sophistication and rigour of the testing will depend upon the target trustworthiness level, and for the higher level there may be a need for additional security or safety related tests.

The 'goto fail' bug suggests there were serious inadequacies in the testing regime Apple applied to the TLS software. Specifically, there should have been a test case for a bad SSL certificate, which would have allowed the development team to test the third condition. If the software failed to detect the bad certificate this should have been investigated. As mentioned in Section 6.1 there also appear to be failures to test code coverage and look for unreachable code.

With protocols like TLS there is clearly a need to ensure that they are robustly engineered and can handle defective or malicious inputs in a trustworthy fashion. The developer of the faulty Heartbeat code is one of the authors of RFC6520 [5] and therefore should have a good understanding of what was required to ensure that the Heartbeat extension was trustworthy, The Heartbleed bug is not one that would have been detected by simplistic pass/fail testing, which could have detected the failure of the Apple code to detect an invalid certificate. The Hearbleed bug required testing that was aimed at validating the robustness of implementation of the Heartbeat protocol. Use of techniques such as fuzz testing might in this case have helped to identify the impact of malformed messages.

6.4 Handling the bugs

The 'goto fail' bug was handled moderately well, but there was scope for improvement in the alerts that were conveyed to the product users. The initial reports suggested that the bug was confined to the iOS product range, i.e. iPhones, iPads and iPods. It soon emerged that the bug also affected other Apple products, most significantly from a cyber security perspective the Mac Book and Mac Server ranges. This led to some confusion over the impact of the bug and the necessary short term measures to reduce the risk. In essence the key issue for many Mac Book users was to avoid use of untrusted public network connections, such as the WiFi networks in public coffee shops, bars, etc. until the OS X patch was available and installed on their Mac Book. From overall business risk, impact and disruption perspectives, this was a serious bug affecting a large number of devices worldwide. Apple deployed a fix for iOS systems on the day the bug was made public and within 4 days thereafter had deployed a patch for the OS X operating system. The fact that the bug only affected Apple products made deployment of the fixes easier, with Apple using its software update channels to push the updates to software end users.

From overall business risk, impact and disruption perspectives the Heartbleed bug was was significantly more serious that the 'goto fail' bug. This is because of OpenSSL's widespread use within the Internet's server infrastructure, its extensive use embedded in products and the diverse nature of organisations that have used the OpenSSL library within their products. From a user perspective, the situation was complicated by the conflicting advice and the initial erroneous labelling of the bug as a virus. Serious code quality issues were subsequently identified in the OpenSSL library raising concerns about the due diligence applied by large companies when they chose to incorporate the library in their products. Given the issues identified by the OpenBSD team, it is unlikely that the code quality of this library had been examined by any of the companies. Three months after the bug was announced, a major manufacturer of industrial control systems was still working on patches

for the OpenSSL vulnerabilities [44]. Since these vulnerabilities can be exploited remotely and there are publicly available exploits, it is not until patches have been developed and users have deployed them on live control systems, that this bug will no longer be an issue.

7 Conclusions

The two bugs discussed in this paper represent major security failures that could, if exploited, cause significant damage. They are good examples of why software trustworthiness matters, and how failure of software developers to adopt trustworthy development practices exposes software users to significant cyber security risks. There is also widespread disruption whilst patches are deployed to affected products and systems. The bugs illustrate how software trustworthiness is an issue for both commercially developed and free/open-source software. The paper examined how the application of three of the TSF principles could have reduced the risk of these bugs being deployed. If the full TSF was applied to the development of this critical software, it is very likely that these bugs would have been spotted and fixed prior to software release.

The confusion during the period after announcement of the Heartbleed bug illustrates the need for authoritative sources of advice that can provide clear advice on appropriate actions or measures consumers and businesses should take. This is an area that should be addressed by the various warning and reporting organisations, e.g. UK-CERT, WARPs, etc.

The apparent failure of companies to exercise due diligence on the quality of software libraries they incorporate into their products is a serious concern. Belatedly a number of these organisations are now providing funding to the OpenSSL to enable investment in their development and testing. This action is welcome and may pave the way to better support for the open source community leading to the development of more trustworthy software.

References

[1] Dierks, T., Rescorla, E. (August 2008). "RFC 5246: The Transport Layer Security (TLS) Protocol, Version 1.2". Available: http://tools.ietf.org/html/rfc5246

[2] Dierks, T., Rescorla, E. (January 1999). "RFC 2246: The Transport Layer Security (TLS) Protocol, Version 1.0". Available: http://tools.ietf.org/html/rfc2246

[3] Dierks, T., Rescorla, E. (April 2006). "RFC 4346: The Transport Layer Security (TLS) Protocol, Version 1.1". Available: http://tools.ietf.org/html/rfc4346

[4] Blake-Wilson, S., Nystrom, M. et. al. (April 2006). "RFC 4366: Transport Layer Security (TLS) Extensions". Available: http://tools.ietf.org/html/rfc4366

[5] Seggelmann, R., Tuexen, M. and Williams, M. (February 2012) "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension". Available: http://tools.ietf.org/html/rfc6520

[6] Apple (2014). ''About the security content of iOS 7.0.6". Available: http://support.apple.com/kb/HT6147

[7] Crowdstrike (2014). "Details about Apple SSL vulnerability and iOS 7.0.6 patch". Available: http://www.crowdstrike.com/blog/details-about-apple-ssl-vulnerability-and-ios-706-patch/index.html

[8] Mitre (2014). "Vulnerability - CVE-2014-1266". Available: http://cve.mitre.org/

[9] Langley, A. (2014). "Apple's SSL/TLS bug (22 Feb 2014)" Available: https://www.imperialviolet.org/2014/02/22/applebug.html

[10] Apple (2014). "Partial listing of sslKeyExchange.c". Available: http://opensource.apple.com/source/Security/Security-55471/libsecurity_ssl/lib/sslKeyExchange.c

[11] Barrett, B. (2014). "Why Apple's Recent Security Flaw Is So Scary". Available" http://gizmodo.com/why-apples-huge-security-flaw-is-so-scary-1529041062

[12] Apple (2014). ''About the security content of OS X Mavericks v10.9.2 and Security Update 2014-001". Available: http://support.apple.com/kb/HT6150

[13] Menn, J. (2014) "Apple security flaw could allow hackers to beat encryption". Available: http://www.reuters.com/article/2014/02/22/us-apple-flaw-idUSBREA1L01Y20140222

[14] Griffiths, S. (2014). "Was YOUR iPhone at risk of being hacked?". Available: http://www.dailymail.co.uk/sciencetech/article-2566675/

[15] Williams, C. (2014). "Update your iThings NOW: Apple splats scary SSL snooping bug in iOS". Available: http://www.theregister.co.uk/2014/02/21/apple_patches_ios_ssl_vulnerability/

[16] Codenomicon (2014). "The Heartbleed Bug". Available: http://heartbleed.com/

[17] Schneier, Bruce (2014). "Heartbleed". Available: https://www.schneier.com/blog/archives/2014/04/heartbleed.html

[18] Curtis, Sophie (2014). "'Heartbleed' bug in web technology threatens user data". The Telegraph. Available: http://www.telegraph.co.uk/technology/internet-security/10754169/Heartbleed-bug-in-web-technology-threatens-user-data.html

[19] Hern, Alex (2014). "Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug". The Guardian. Available: http://www.theguardian.com/technology/2014/apr/08/heartbleed-bug-puts-encryption-at-risk-for-hundreds-of-thousands-of-servers

[20] Steinberg, J. (2014). "Massive Internet Security Vulnerability – Here's What You Need To Do". Forbes. Available: http://www.forbes.com/sites/josephsteinberg/2014/04/10/massive-internet-security-vulnerability-you-are-at-risk-what-you-need-to-do/

[21] Prigg, M. (2014). "Major security alert over 'heartbleed' eavesdropping bug that could have infected TWO THIRDS of sites". Available: http://www.dailymail.co.uk/sciencetech/article-2599896/

[22] Woollaston, V. (2014) "Are YOUR details at risk from 'heartbleed' hackers?". Available: http://www.dailymail.co.uk/sciencetech/article-2600701/

[23] Evans, R. and Steere, T. (2014). "Internet users told to change ALL passwords in security alert over 'catastrophic' Heartbleed bug". http://www.dailymail.co.uk/sciencetech/article-2601096/

[24] Hern, Alex (2014). "Heartbleed: don't rush to update passwords, security experts warn". The Guardian. Available: http://www.theguardian.com/technology/2014/apr/09/heartbleed-dont-rush-to-update-passwords-security-experts-warn

[25] Curtis, Sophie (2014). "Heartbleed bug: which passwords should you change?". The Telegraph. Available: http://www.telegraph.co.uk/technology/internet-security/10756807/Heartbleed-bug-which-passwords-should-you-change.html

[26] Gibbs, Samuel (2014). "Heartbleed bug: what do you actually need to do to stay secure?". The Guardian. Available: http://www.theguardian.com/technology/2014/apr/10/heartbleed-bug-everything-you-need-to-know-to-stay-secure

[27] Albert, M. (2014). "Heartbleed virus: Changing your password may not eliminate risk". CBS Evening News. Available: http://www.cbsnews.com/videos/heartbleed-virus-changing-your-password-may-not-eliminate-risk/.

[28] MUMSNET (2014), "Mumsnet and Heartbleed as it happened". Available: http://www.mumsnet.com/features/mumsnet-and-heartbleed-as-it-happened

[29] Mitre (2014). "CVE-2014-0160". Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160

[30] Cassidy, S. (2014) "Diagnosis of the OpenSSL Heartbleed Bug". Available: http://blog.existentialize.com/diagnosis-of-the-openssl-heartbleed-bug.html

[31] Cisco Security Advisory (2014). "OpenSSL Heartbeat Extension Vulnerability in Multiple Cisco Products". Available: http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20140409-heartbleed

[32] OpenVPN (2014). "OpenSSL vulnerability - Heartbleed". Available: https://community.openvpn.net/openvpn/wiki/heartbleed

[33] Hern, Alex (2014). "BT preparing to fix Heartbleed-affected Home Hubs". The Guardian. Available: http://www.theguardian.com/technology/2014/apr/16/bt-heartbleed-home-hubs

[34] ICS-CERT (2014). "Alert (ICS-ALERT-14-099-01E)". Available: http://ics-cert.us-cert.gov/alerts/ICS-ALERT-14-099-01E

[35] TSI (2014). "About". Available: http://www.uk-tsi.org

[36] BSI (2014). "PAS 754:2014 Software Trustworthiness. Governance and management. Specification". BSI, London.

[37] van Deursen, A (2014). "Learning from Apple's #gotofail Security Bug". Available http://avandeursen.com/2014/02/22/gotofail-security/

[38] LibreSSL (2014). "LibreSSL". Available: http://www.libressl.org/

[39] Seltzer, L. (2014). "OpenBSD forks, prunes, fixes OpenSSL". Available; http://www.zdnet.com/openbsd-forks-prunes-fixes-openssl-7000028613/

[40] Barr, M. (2014). "Apple's #gotofail Security Bug was Easily Preventable". Available: http://embeddedgurus.com/barr-code/2014/03/apples-gotofail-ssl-security-bug-was-easily-preventable/.

[41] McAllister, N. (2014). "OpenBSD founder wants to bin buggy OpenSSL library, launches fork". The Register. Available http://www.theregister.co.uk/2014/04/22/openssl_fork_libressl/

[42] Mitre (2014). "CWE-125: Out-of-bounds Read". Available: http://cwe.mitre.org/data/definitions/125.html

[43] Mitre (2014). "CWE-126: Buffer Over-read". Available: http://cwe.mitre.org/data/definitions/126.html

[44] ICS-CERT (2014) . "Siemens OpenSSL Vulnerabilities". Available: http://ics-cert.us-cert.gov/advisories/ICSA-14-198-03

Note: All websites were last accessed: 21 July 2014.