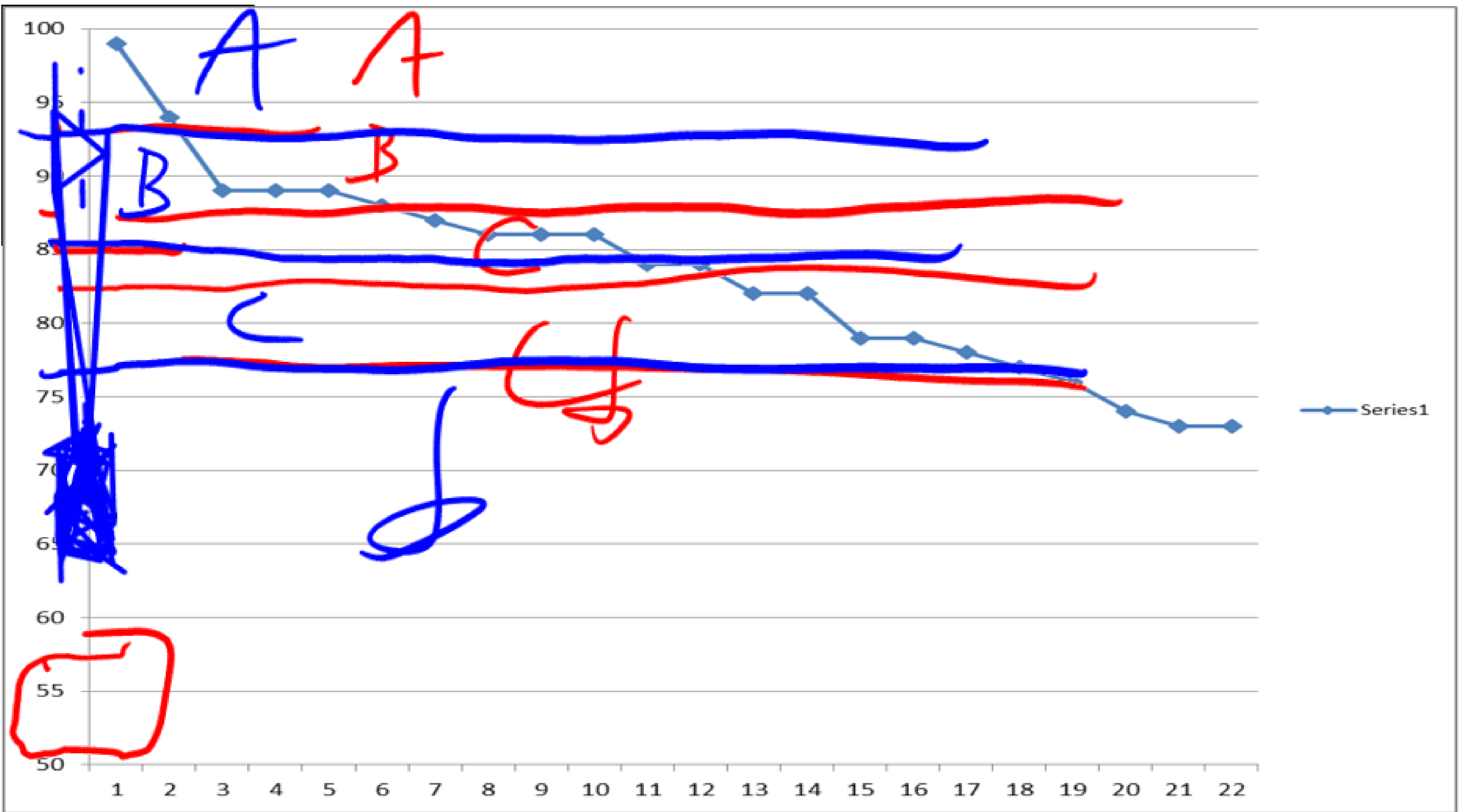


83.4%	Average	84.4%	95.5%	96.4%	90.0%	87.3%	70.9%	70.5%	73.6%
84.0%	Median	88.0%	100.0%	100.0%	100.0%	100.0%	100.0%	80.0%	80.0%
6.8%	STD	8.3%	11.8%	10.0%	17.2%	28.6%	38.4%	30.2%	17.1%
73.0%	Min	60.0%	50.0%	60.0%	40.0%	0.0%	0.0%	0.0%	20.0%
99.0%	Max	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Secure Software Development Buffer Overflow

Objectives

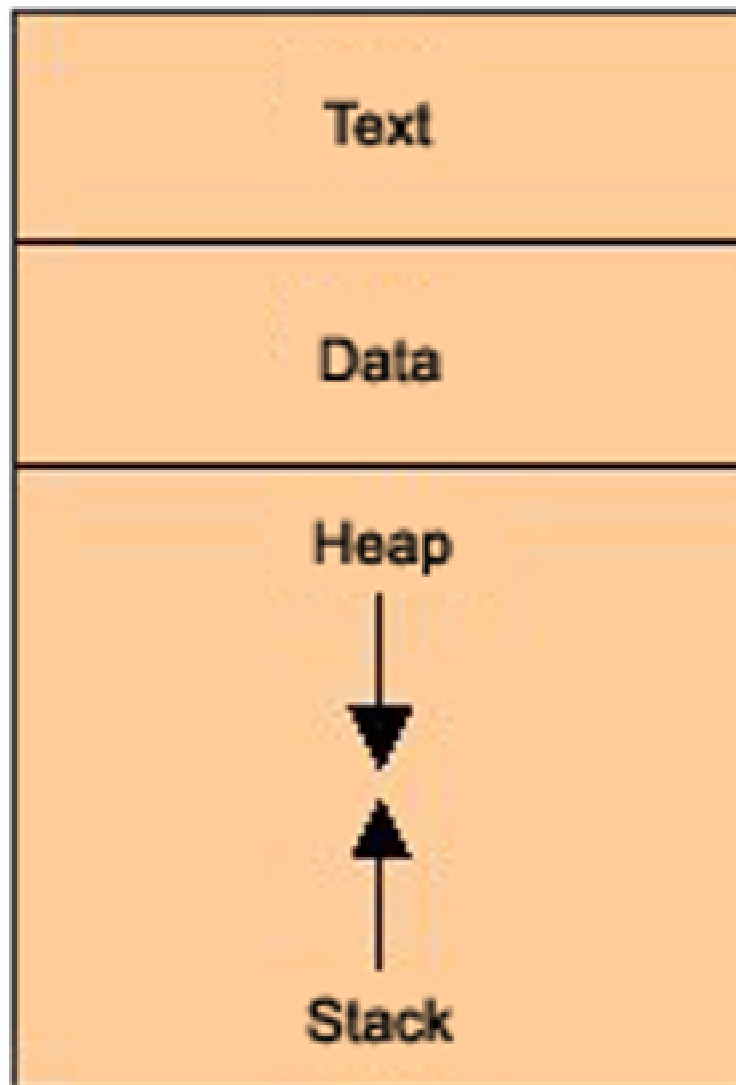
- Explain the concept of variadic functions
- Recognize the causes of buffer overflows
- Recognize instances of buffer overflow within source code
- Explain how one can exploit a buffer overflow
- List mechanisms which can be used to help prevent buffer overflows

What causes buffer overflow?

- Occurs anytime the program writes more information into the buffer than the space it has allocated in the memory.
  - Allows an attacker to overwrite data that controls the program execution path and hijack the control of the program to execute the attacker's code instead the process code.

MSOE

Low Addresses



High Addresses

MSOE

Buffer Overflows

- Description
  - Accessing locations of a buffer outside the boundaries of the buffer
- Common causes
  - C-style strings
  - Array access and pointer arithmetic in languages without bounds checking
  - Off by one errors
  - Fixed large buffer sizes (make it big and hope)
  - Decoupled buffer pointer and its size
    - If size unknown overflows are impossible to detect
    - Require synchronization between the two
    - Ok if size is implicitly known and every use knows it (hard)

MSOE

```
#include

char *code = "AAAABBBBCCCCDDD"; //including the
   character '\0' size = 16 bytes

void main()

{

  char buf[8];

  strcpy(buf, code);

}
```


MSOE

Steps to exploit a buffer overrun.

- 1. Discovering a code, which is vulnerable to a buffer overflow.
- 2. Determining the number of bytes to be long enough to overwrite the return address.
- 3. Calculating the address to point the alternate code.
- 4. Writing the code to be executed.
- 5. Linking everything together and testing

MS OE

C Language Information

- What does this line of code do?
- Printf(“%p\n”)

MSOE

Lets work through a demo

- Lets look at the call tree

MSOE

Preventing Buffer Overflows

- Secure coding practices
  - Using the best approaches we know to prevent the injection of buffer overflows into the code base.
- Safe library implementations
  - library-based defenses that use re-implemented unsafe functions and ensure that these functions can never exceed the buffer size.
    - IE Libsafe project.
- Stack protection
  - Software which monitors the stack and emits an alert if a stack corruption is detected
    - the SecureStack developed by SecureWave.
- Compiler based runtime boundaries
  - Compilers inject boundaries into the executtable
- Static Analysis

MSOE

ORACLE®

CON7402 – ‘Heartbleed’ (CVE-2014-0160) Case Study I

Secure Development Perspective

John Heimann – Vice President, Security Program Management
Eric Maurice – Director, Oracle Security Assurance

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved. |

Program Agenda

1. What is the '*Heartbleed*' SSL vulnerability?
2. How did it come about?
3. What are the lessons learned?
4. What is the implication for open source development?
5. What lessons can YOU derive for YOUR organization?

Introduction

- Today's discussion came about as a unique opportunity
- We believe that any organizations can derive significant lessons from 'Heartbleed':
  - Secure development practices
  - Security testing
  - Software acquisition and use of open source components
  - Security responses policies and procedures
- This presentation is the first of a 2-part session:
  - Heartbleed case study I: Secure Development Perspective - **NOW**
  - Heartbleed case study II: Vulnerability Handling Perspective – **Thursday, 2:30PM**

MSOE

Program Agenda

1. What is the '*Heartbleed*' SSL vulnerability?
2. How did it come about?
3. What are the lessons learned?
4. What is the implication for open source development?
5. What lessons can YOU derive for YOUR organization?

CVE-2014-0160 a.k.a. Heartbleed

What is it?

- A vulnerability affecting certain versions of the OpenSSL crypto libraries:
  - OpenSSL 1.0.1 through 1.0.1f (inclusive) are vulnerable
- A successful exploitation can result in allowing malicious attacker with the ability to remotely (over the Internet) read (sections of) the memory of the targeted system:
  - Possible compromise of secret keys and other sensitive information
- It was called 'Heartbleed' because the bug originated in the OpenSSL's implementation of the TLS/DTLS (transport layer security protocols) heartbeat extension

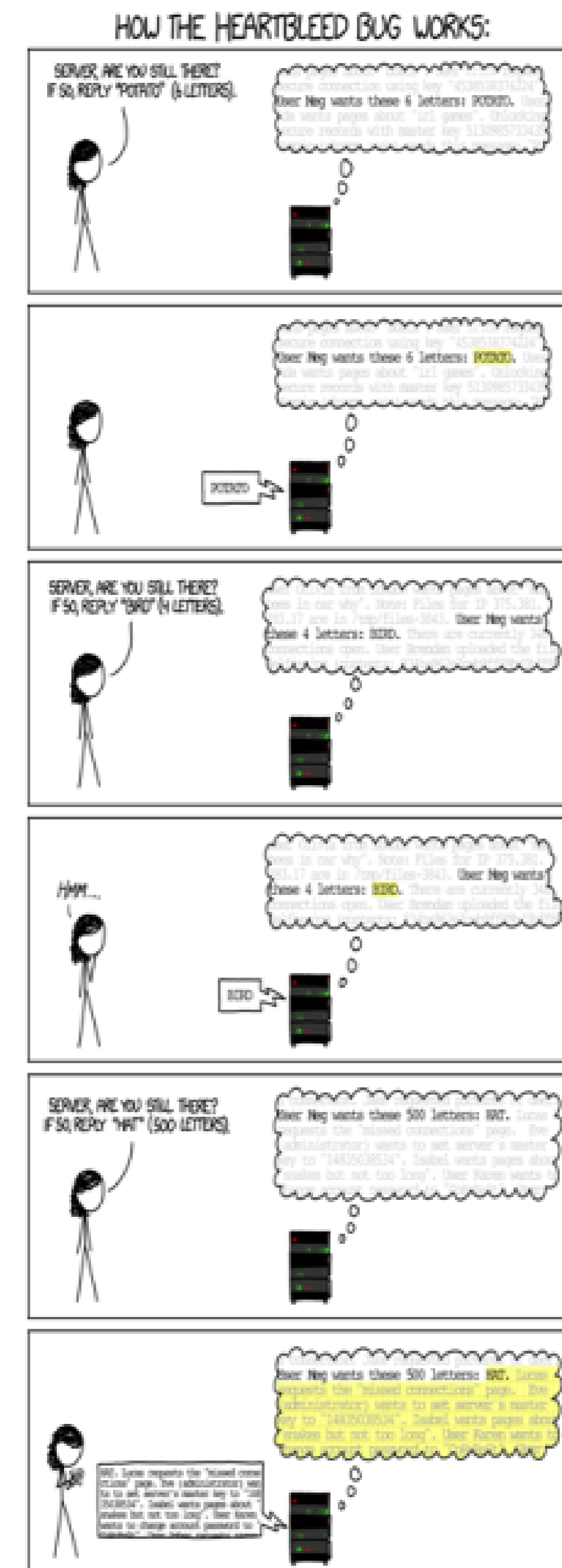
Lets look at some code.

- http://nakedsecurity.sophos.com/2014/04/08/anatomy-of-a-data-leak-bug-openssl-heartbleed/

This vulnerability was a big deal...

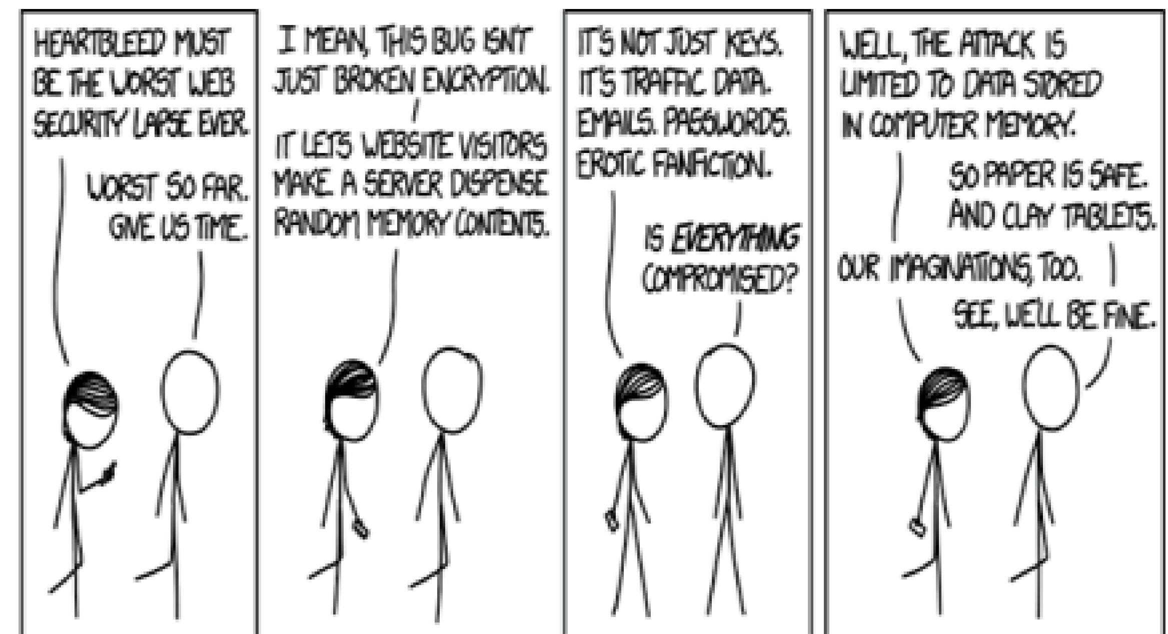
- Very prevalent use of OpenSSL – the open source toolkit implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security (TLS v1)
- These are cryptographic protocols intended to secure communications over the Internet
  - Web browsing
  - Emails
  - VoIP
  - Etc.

Source:
http://xkcd.com/1354/



… But the world didn't come to an end!

- Tremendous visibility in the press and security community resulted in quick response
- The vulnerability was limited to certain versions of OpenSSL
- The common use of SSL termination proxy (particularly with large sites) provided some level of mitigation against successful exploitation



Source: http://xkcd.com/1353/

Program Agenda

1. What is the '*Heartbleed*' SSL vulnerability?
2. How did it come about?
3. What are the lessons learned?
4. What is the implication for open source development?
5. What lessons can YOU derive for YOUR organization?

"In one of the new features, unfortunately, I missed validating a variable containing a length… In this case, it was a simple programming error in a new feature, which unfortunately occurred in a security relevant area"

– Robin Seggelmann

(Source: Sidney Morning Herald)

How was it discovered?

- Google and Codenomicon independently reported Heartbleed in April 2014
- OpenSSL may have been discovered and exploited by others who did not report it
  - OpenSSL code is open source and available for review by both well-intentioned and malicious analysts
  - A researcher review of audit logs suggested that hackers may have been aware of and may have exploited Heartbleed for at least five months prior to April 2014
- Hackers were definitely exploiting Heartbleed within days after it was reported

MSOE

Program Agenda

1. What is the '*Heartbleed*' SSL vulnerability?
2. How did it come about?
3. What are the lessons learned?
4. What is the implication for open source development?
5. What lessons can YOU derive for YOUR organization?

Thou shall not trust user input

- User input may
  - Have the wrong value
  - Have the wrong size or format
  - Be from an authorized but malicious user
  - Be from an unauthorized user (or have been modified by an malicious user)
- User input must be validated in form and content
  - Security functions should never rely on unvalidated user input
  - Malicious input to functions which aren't themselves security related can still cause security failures

MSOE

Vulnerabilities due to unvalidated user input

- Buffer overflow
- Integer overflow
- SQL Injection
- Character encoding attacks
- Cross site scripting
- Cross site request forgery
- …
- Heartbleed

← Risk | 2013 Table of Contents | A1-Injection →

2013 Top 10 List

A1-Injection	Injection flaws, such as SQL, OS, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
A2-Broken Authentication and Session Management	Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities.
A3-Cross-Site Scripting (XSS)	XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation or escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.
A4-Insecure Direct Object References	A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.
A5-Security Misconfiguration	Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.
A6-Sensitive Data Exposure	Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.
A7-Missing Function Level Access Control	Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.
A8-Cross-Site Request Forgery (CSRF)	A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.
A9-Using Components with Known Vulnerabilities	Components, such as libraries, frameworks, and other software modules, almost always run with full privileges. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications using components with known vulnerabilities may undermine application defenses and enable a range of possible attacks and impacts.
A10-Unvalidated Redirects and Forwards	Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Source: 2013 OWASP Top 10

MSOE

Program Agenda

1. What is the '*Heartbleed*' SSL vulnerability?
2. How did it come about?
3. What are the lessons learned?
4. What is the implication for open source development?
5. What lessons can YOU derive for YOUR organization?

Questioning closely-held assumptions

- 1,000,000 eyes, really?
  - Quality of code is directly related to the skills of the programmers
  - Code review doesn't happen automatically
    - Publication of code doesn't mean it is getting reviewed
    - What are the skills of the reviewers?
  - Security assurance requires time, effort and money (tools, experts, etc.)
- Open source vs. security by obscurity
  - Empowerment of hackers through public visibility into the code?
  - Unique challenges of coordinated disclosure with open source components
- It's really ALL ABOUT the level of trust you have in the producer(s) of the code

MSOE

Open Source *can* be High Assurance

- Requires investment in security to make it secure
- MySQL, Oracle Linux, Open Solaris, and OracleVM are open source
  - Development and support teams are committed the security of the product
  - Follow the same OSSA security practices that all Oracle products are required to follow

MSOE

Java Standard Edition (SE) is an interesting case study

- Prior to Oracle's acquisition of Sun, Java SE relied heavily on the "million eyes" of the community for security
  - Some vulnerabilities were found and fixed through the community process
  - Many more were found by professional researchers, and made public after Oracle's acquisition of Java
- Since Oracle's acquisition, Java has adopted Oracle security practices
  - Java development process has much greater pre-release security analysis and testing
  - Java security architecture has been improved
  - Better tools to manage Java security and remove old, vulnerable versions Java
  - *Significantly fewer vulnerabilities have been reported in Java*
- Java is still open source, but security improvements have come from Oracle's active investment in Java security

MSOE

Program Agenda

1. What is the '*Heartbleed*' SSL vulnerability?
2. How did it come about?
3. What are the lessons learned?
4. What is the implication for open source development?
5. What lessons can YOU derive for YOUR organization?

Key take-aways

1. Know where your code is coming from, and…
2. Assess the security assurance practices associated with anything used in your environment
   - COTS, open source, internally developed applications, etc.
3. Strong security assurance practices should include
   - Implementation of core security standards
   - Security testing
   - Effective security vulnerability remediation procedures

MSOE

ORACLE®

ORACLE®

MSOE