Analyzing Code for Security Defects

How to review large code base
for security defects.

Security
compass

AGENDA

- Background
- Methodology
  - Threat Analysis
  - Assign Value to Threat (Propose a different technique)
  - Common List of Issues (Interactive)
- Questions

Security compass

Defense In Depth Past & Present

- Firewalls
- Separation of Networks (DMZ)
- Network / Host Assessments
- Bastion Hosts / Hardened Builds
- Managed Vulnerability Scanning
- Product Review/Application Assessment
- **Code Review**

Security compass

Code Reviews

- Automated Tools
  - Static Code Scanners
    Example: RATS, ITS4 etc.
  - Compile Time and Run Time Scanners
    Example: Ounce Labs, Secure Software, GCC patch etc.
- Manual Auditing
  - Small Code base (Not a Problem)
  - Large Code base (Big Problems)

Security compass

Methodology of Reviewing Large Code Base

1. Threat Model
2. Cursory Review of Code
3. Separation of Code [Standard Model & Application Architecture]
4. Maintain code notes with reviewer name
5. Detailed Code Analysis
6. Common list of issues to review **[**C/C++ Language Specific**]**

Security compass

Threat Analysis

- Overview
- What Is Threat Analysis
- When Threat Analysis
- Why Threat Analysis
- Who Threat Analysis
- How Threat Analysis
  - Collecting Information / Decomposing Application
  - Modeling the System
  - Analysis to Determining Threat

Security compass

Common List of Issues: C/C++ Specific

- Termination
  - Null Termination
  - Conditional Termination
  - Premature Termination
- Validation
  - Exported Functions
  - Command Line
  - String Formatting
- Calculations
  - Division
  - Signed
  - Integer
  - Off by one / few

Just A Few

Security compass

Threat Modeling

Security compass

What is Threat Modeling ?

- Threat modeling is an organized method of attacking an application. It can be considered as a systematic method of finding security issues in application.
- Threat Modeling can be viewed as a reversal of roles, where by a developer attempts to think as an attacker to determine possible compromises/threats in his application.

Security compass

What is Threat Modeling ?

- Hackers/Attackers have been threat modeling for a while now "Brainstorming". They haven't used the terminology "Threat Modeling"
- Security Groups at software houses have formalized the process to help developers and testers better understand the different threats that might exists in an application.

Security compass

Why Threat Model ?

Threat Modeling can help –

- Develop countermeasures for threats identified
- Weigh each threat (assign value to them)
- Produce a secure application
- Review code for security defects in large code base / Binary Analysis
- Understand threats to the application [Developers to Business Owner]

Security compass

Who should Threat Model ?

- Developers / PM / Business Dev /Security Group / Any One else ....
- Business Dev
  - Explain the goal of the application. (so the main goal is still met).
- PM / Application Architect
  - Provide Data Flow Diagram / Application architecture and explain the app path in detail
  - Help understand why a particular path is chosen to develop the application
- Developers
  - Approximate time frame on the application dev process
  - Understand potential threats.
- Security Group
  - Point out different points of weakness (Risks & Threats).

Security compass

When should you threat Model ?

» Most suitable to perform threat model after the application architecture has been developed (After design stage)

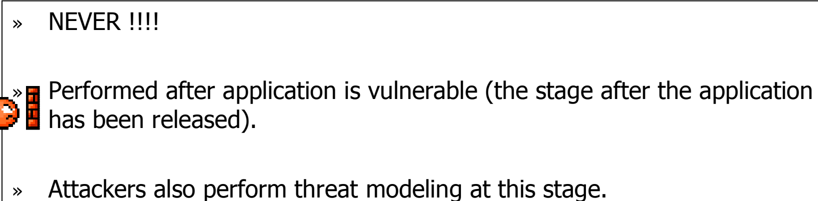
» Threat Modeling must be revisited at least once when the application is in Alpha release (Before formal testing starts).

» Ideally Threat Modeling must be performed every time an application is tested for any change (Functionality/Security/Any other fix/Upgrade).

Security compass

When do you typically threat Model ?

» NEVER !!!!



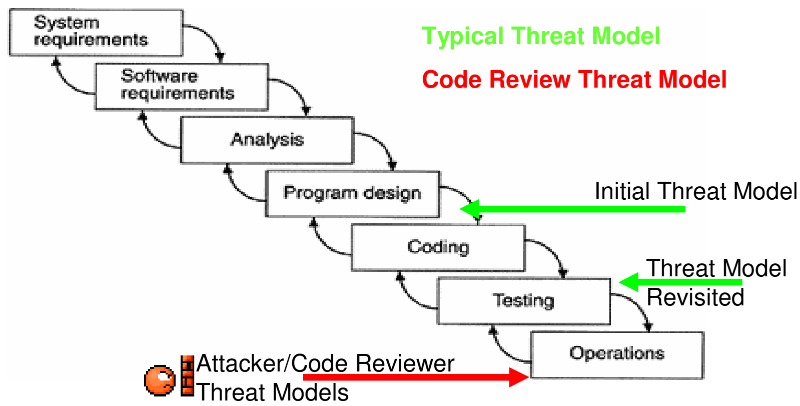
» Performed after application is vulnerable (the stage after the application has been released).

» Attackers also perform threat modeling at this stage.

Note: Every application that is being developed or has already been developed should be threat modeled, even if the application is being built for internal use only.

Security compass

SDLC - Waterfall Model



HackInTheBox 2005 Securitycompass.com

Security compass

So Far we have seen

» What Is Threat Analysis
» When Threat Analysis
» Why Threat Analysis
» Who Threat Analysis

Lets see how to perform Threat Analysis (High Level)

HackInTheBox 2005 Securitycompass.com

Security compass

How to Threat Model (The Process)

» Step 1 - Collecting Information about the application (background why the application is built etc).

» Step 2 - Decomposing Application / Modeling the System (break the application down into reasonably separate chunks either by functionality or connectivity).

» Step 3 - Analysis to Determine Threats (perform a walk through to determine the different locations of issues).

Security compass

Step 1

Collecting Information

Collecting Information

- How the application is intended or not intended to be used in deployment
- Any dependencies that exists (external / inter process dependencies / account level dependencies / application requirement example: mail server etc).

Security compass

Step 2

Decomposing

Decomposing Application / Separating the application into reasonable chunks

- APPLICATION ARCHITECTURE: Chunks either based on Application Architecture / functionality (specially if performing code review, help code reviewer).

AND/OR Preferably BOTH

- INDIVIDUAL COMPONENTS: Chunks based on individual components (Entry Points, Trust Points etc)

Security compass

Step 2a

Decomposing – Individual Components

- Entry Points
  - Identify all entry points to the application
  - Network accessible (RPC / TCP /Web Services etc)
  - Locally accessible (Registry / LPC / File / command line / environment variables etc)
- Trust Level
  - Identify different trust boundaries

Step 2a Contd.

Security compass

Decomposing – Individual Components

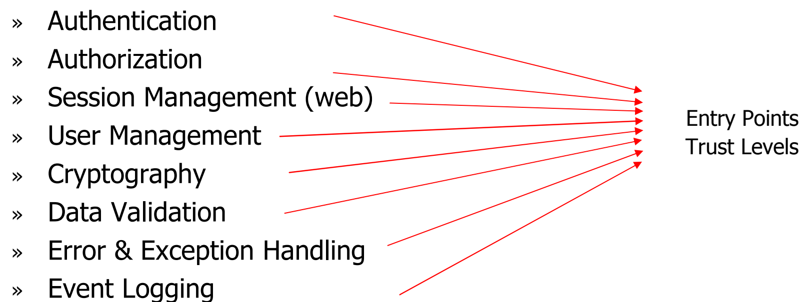
- Data Flow Diagrams
  - Drawing data flow diagrams or other models to visual represent the application is called modeling the system.
- A DFD
  - A graphical representation showing communication between objects
  - Describe activities that process data
  - Show how data flows through a system
  - Show logical sequence of associations and activities

Step 2b

Security compass

Decomposing - Application Architecture

- Authentication
- Authorization
- Session Management (web)
- User Management
- Cryptography
- Data Validation
- Error & Exception Handling
- Event Logging



Security compass

Step 3

Analysis to Determine Threats

- Identify threats and create attack scenarios on the basis of the DFD (is the single biggest challenge).
- Analysis of threats can determine if a threat is mitigated or can result in a vulnerability.
- Threats is not the same as vulnerabilities:-
  - Threats – possibly dangerous
  - Vulnerability – susceptible to attack (Vuln could be Known or Unknown)
- Unmitigated threat turns into a vulnerability - WSC2

Security compass

Definitions

Dictionary.com

- **Threats**: One that is regarded as a possible **danger**; a menace.
- **Vulnerabilities**: Susceptible to **attack**.

Writing Secure Code 2

- **Threats**: A malicious entity that might try to attack. A Threat does not **constitute** a vulnerability.
- **Vulnerabilities**: A weakness in a system that can be exploited. A Vulnerability exists when there is a Threat that goes unmitigated.

Security compass

Assigning Value to Threat

HackInTheBox 2005

Securitycompass.com

Security compass

Assigning Value to Threats.

» The second biggest challenge is assigning value to threats.
(Note: First was Identify threats and create attack scenarios on the basis of the DFD)

» A Model used and developed at MS is the DREAD model.
DREAD (WSC) (1-10) / 5
- Damage Potential
- Reproducibility
- Exploitability
- Affected Users
- Discoverability

HackInTheBox 2005

Securitycompass.com

Security compass

Application Architecture – Threat Model

- Authentication
- Authorization
- Cryptography
- Data Validation
- Error & Exception Handling
- Logging

(Low-1, Mid-2, High-3)
Low (Intranet)
Mid (Internet Non Critical – no PII or other critical data is being pulled)
High (Internet Critical Data)

Propose Technique

HackInTheBox 2005 Securitycompass.com

Security compass

Application Architecture – Threat Model

- Modeling the System
  - Entry Points in each location
  - Trust Levels
- Assign Value Depending on location of each Architecture
  - Authentication 1 2 3
  - Authorization 1 2 3
  - Crypto 1 2 3
  - Data Validation 1 2 3
  - Error 1 2 3
  - Logging 1 2 3

Propose Technique

HackInTheBox 2005 Securitycompass.com

Application Architecture – Threat Model

Propose Technique

- Min is > 8 Low Risk
- Mid if > 12 Medium Risk
- Max if > 18 High Risk

- Total if within range then considered high risk (13-18) then **review code for sure**, if bug found then it should follow best security practice and **fix bugs immediately**.
- Total if within range then considered Medium risk (7-12) then **try to review code**, if bug found then should follow best security practice and need to fix bugs at **reasonable quick**. (**next patch release**)
- Total if within range then considered low risk (>6) then **next time around review code**, if bug found should follow best security practice and need to fix bugs. (**next point release**)

Threat Model Checklist

- ✓Every Application Should Have A Threat Model At Least Once
- ✓Every Threat Must Be Analyzed
- ✓A Threat Value Must Be Assigned to Every Threat
- ✓Bugs Must Be Fixed Based On The Threat Value

Security compass

How Do You Review Code?

Security compass

Methodology

- ☑ **Step 1) Threat Model**
- ☑ **Step 2) Everyone Read The Code**
- ☑ **Step 3) Break Code Into Separate Chunks / Same as DFD**
- ☑ **Step 4) Maintain code notes with reviewer name**
- ☑ **Step 5) Detailed Code Analysis**

Step 6) Common list of issues to review [C/C++ Language Specific]

HackInTheBox 2005 Securitycompass.com

Security compass

Reviewing Code

Security compass

C/C++ Some of the Common Issues

» Commonly seen issues while reviewing code.

» Many Issues Exists, we will cover three such topics, which lead to vulnerabilities in applications.

- Termination Issues
- Validation Issues
- Calculation Issues

Security compass

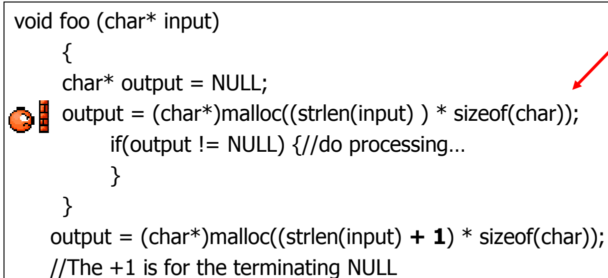
» Termination

- Null Termination and strlen
- Conditional Termination
- Premature Termination

Security compass

Null Termination "\0" and strlen

Hint: strlen



```
void foo (char* input)
    {
    char* output = NULL;
    output = (char*)malloc((strlen(input) ) * sizeof(char));
        if(output != NULL) {//do processing…
        }
    }
  output = (char*)malloc((strlen(input) + 1) * sizeof(char));
  //The +1 is for the terminating NULL
```

When allocating memory for a string always bear in mind the fact that the ***strlen* returns the length of a string excluding the terminating NULL**. Hence, it is necessary to explicitly allocate for this terminator as shown.

Security compass

NULL Termination "\0" and strlen

- If strlen is not increased by one then, string operations would not perform as expected.
- When copying string characters manually in a loop, it is important to NULL terminate them at the end. This issues is seen when a programmer attempts to handle the length properly, however, a string can be created without a trailing NULL. This often happens when using a *strncpy* type function operation.

Security compass

Conditional Termination

```
{   int StringLength;
    size_t index;
    int BufferLength=20;
    TCHAR *Buffer=argv[1];
    index = strlen(Buffer);
    printf ("%d", index);
    while (index < BufferLength && Buffer[index] != '\0')
          index++;
    StringLength = strlen(Buffer);
    }
```

Hint: If index = 5 Then what happens?

first clause fulfilled termination condition, strlen reads past true end of buffer

Conditional Termination

» The loop seems to be attempting to check that the buffer is properly NULL-terminated without overflowing the end of the buffer, but the statement immediately following assumes that the terminator was found, and thus the second condition is what terminated the while loop.

» However, if the first clause is what fulfilled the termination condition, the strlen call will read past the true end of the buffer.

» It is therefore important to ensure that the logic checks for all conditions including failures.

Premature Termination

```
int main(int argc, char* argv[])
{
    int a = 1;
    if (a==2);
    {
        printf ("hello world\n");
    }
    return 0;
}
```

; terminates statements

Security compass

Premature Termination

» In the C and C++ programming languages, **;** terminates statements

» Premature termination of the if-statement causes the subsequent statement to be executed always.

Security compass

» Validation

- Exported Functions
- Command Line
- String Formatting

Validation – Command Line

```
int copy(char* input) {
      char var[20];
      strcpy (var, input);
      return 0;
}

int main(int argc, char* argv[]){
      copy(argv[1]);
      return 0;
}
```

HackInTheBox 2005

Securitycompass.com

Validation – Reading from network

```
void pr( char *str)
{    //buf max limit to 2000
   char buf[2000]="";
   strcpy(buf,str);
}
while( bytesRecv == SOCKET_ERROR )
//receive the data that is being sent by the client max limit to 5000 bytes.
{   bytesRecv = recv( clientSocket, Message, 5000, 0 );
   if ( bytesRecv == 0 || bytesRecv == WSAECONNRESET )
         break;
    pr(Message);
   ........
```

HackInTheBox 2005

Securitycompass.com

Security compass

Validation – Exported Functions

» Any public function, for example, an exported function from a dynamic or statically linked library or a function accessible via an RPC interface, is vulnerable to attack via it's parameters.

```
//filenam is an input filename, len is the length
__declspec(dllexport) int expfunc (char *Filenam, size_t Len)
  {
    char szCopy[MAX_PATH]; //260
    strncpy(szCopy, Filenam, Len);
    return 0;
  }
```

Hint: If len = 500 Then what happens?

» In the example **if Len is greater than MAX_PATH then** Copy will not be large enough to accommodate the data being copied. All public functions should always validate all the input passed to them.

HackInTheBox 2005

Securitycompass.com

Security compass

Validation – String Formatting

```
void main()
{
    char buf[20]="";
    strncpy(buf,argv[1],20);
    printf(buf);
}
```

The function prints the data that is provided as an argument to the function using printf function.

The function however **doesn't format the data**. what would happen if argv[1] contained "%.x"

HackInTheBox 2005

Securitycompass.com

Security compass

» Calculation

- Division
- Signed
- Integer
- Unicode
- Off by one / few

Security compass

Calculation - Division

```
int divide(long x)
{   long y;
    y = (4096 / (x / sizeof(long)));
    printf("%d\n", y);
    return y;}
int main(int argc, char* argv[])
{   int x;
    sscanf(argv[1], "%d", &x);
    divide(x);
    return 0;
}
```

Hint: What is the lowest value of x

X=1,2,3 then division by zero, since integer division of ¼ results in 0

Security compass

Calculation - Division

» If the value of *x* is larger than the *sizeof(long)* (usually 4 bytes) the program would function properly, however when *x* is less than 4, for instance 1, the value of r is (4096) / (¼) which would result in division by zero, since integer division of ¼ results in 0.

» Hence, special care should be taken in algorithms that require calculation to be performed with either user supplied variables or derivatives of user supplied variables to ensure that there is no possibility of division by zero.

» While performing division on any values, ensure that the division is checked, even if the caller is a trusted source.

Security compass

Calculation - Integer

```
int main(int argc, char **argv)
{   int i;
    u_int malloc_size, size;
    char *data;
    size = atoi(argv[1]);
    malloc_size = size * 4;
    data = (char *)malloc(malloc_size)
    if(data != NULL)
        { for(i = 0; i < size; i++)
          data[i] = argv[2][i];
        …….
```

u_int=maxvalue of (int)

Hint: what is the data type of size compare it to malloc_size (u_int)

Security compass

Calculation - Integer

» The data type of a variable defines the maximum / minimum value allowed for it, based on the number of bytes it occupies.

» What a user can do is pass to the program a large integer. When the program calculates: *malloc_size = size * 4*, the *malloc_size* variable will be overflowed and truncated.

» For instance, if a variable is declared as *short*, the maximum value the variable can store is 32767 (16 bits or 2 bytes long) while the minimum value is -32767. Hence, if the value stored exceeds 32767 it would lead to corruption of data.

Security compass

Calculation - Signed

```
int main(int argc, char **argv)
{
    int size;
    char data[1024];
    size = atoi(argv[1]);
    if(size > 1024)
            return;
    memcpy(data, argv[2], size);
}
```

Hint: what happens if you provide maxint+1

Security compass

Calculation - Signed

» The example attempts to implement a check to prevent integer overflow - *if (size > 1024)* -, however there is a subtle flaw, in that it uses a signed integer for the size variable. Thus, the check can be defeated by specifying a negative number. Ensuring the right data type is used could prevent this error e.g. u_int size;.

» Signed issues occur when a signed variable is interpreted as an unsigned variable. This commonly occurs in cases where casting is used to convert from signed to unsigned types and vice versa.

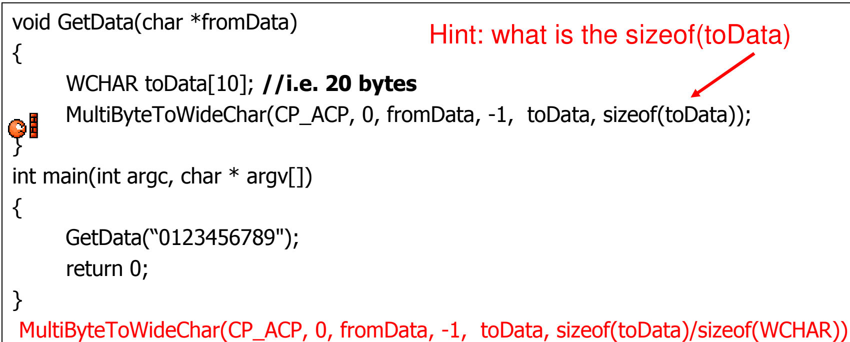
» While creating loops as a best practice always use unsigned integers.

HackInTheBox 2005

Securitycompass.com

Security compass

Calculation: Unicode



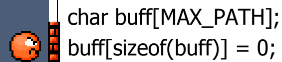
```
void GetData(char *fromData)
{
      WCHAR toData[10]; //i.e. 20 bytes
      MultiByteToWideChar(CP_ACP, 0, fromData, -1,  toData, sizeof(toData));
}
int main(int argc, char * argv[])
{
      GetData("0123456789");
      return 0;
}
MultiByteToWideChar(CP_ACP, 0, fromData, -1,  toData, sizeof(toData)/sizeof(WCHAR))
```


HackInTheBox 2005

Securitycompass.com

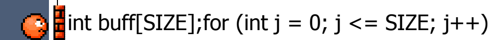
Calculation – Off by one/few

» C/C++ arrays start at 0



```
char buff[MAX_PATH];
buff[sizeof(buff)] = 0;
```

» **should be buff[sizeof(buff) -1]**



```
int buff[SIZE];for (int j = 0; j <= SIZE; j++)
```

» **should be < SIZE and not <= buff[j] = 0;**

Methodology

- ☑ Step 1) Threat Model
- ☑ Step 2) Everyone Read The Code
  (Cursory review to understand contents of each file and global vars)
- ☑ Step 3) Break Code Into Separate Chunks / Same as DFD
  (Individual can review their sections)
- ☑ Step 4) Maintain code notes with reviewer name
- ☑ Step 5) Detailed Code Analysis
- ☑ Step 6) Common list of issues to review [C/C++ Language Specific]

Security compass

Methodology

- ✓Threat Model
- ✓Do a Two Pass Code Review
- ✓Break Code Review Into Major Separate Sections (As in TM)
- ✓Create Major Worksheet Which Consists Of Global Variables, Global Functions and Classes.
- ✓Code Reviewer Should Append His Own Comments About The Code.

HackInTheBox 2005 Securitycompass.com

Security compass

Questions

Nish[a t]securityCompass.com
Visit us at
www.SecurityCompass.Com

HackInTheBox 2005 Securitycompass.com