COMP8050 – Security for Software Systems

Dr. David Stynes

Software Vulnerabilities Fundamentals

Dr. David Stynes

Vulnerabilities vs Bugs

- Vuln: a specific flaw or oversight that allows attackers to do something malicious
- Bug: an error, mistake or oversight that results in an unexpected and typically undesirable behaviour
- Exploiting: attacking a vuln

Vulnerability Definition

From Bruce Schneier (https://www.schneier.com/) :

- Vulnerabilities are software mistakes in specification and design, but mostly mistakes in programming.
- Any large software package will have thousands of mistakes.
- Once discovered, they can be used to attack systems.
- This is the point of security patching: eliminating known vulnerabilities.
- But many systems don't get patched, so the Internet is filled with known, exploitable vulnerabilities.

Finding Vulnerability

Application Penetration Testing

Fuzzing

Reverse Engineering

Source Code Review

Or.. Being more advanced:

- Tracking software bugs that might get patched in the future
- Reversing security patches
- Some of the patches might themselves be vulnerable, or might not be applied to all systems

Who wants to find Vulnerability

Academic Researchers

Hackers

Software Companies

Criminals

Governments

Media – to write about security issues

How much is a 0-day Vulnerability worth?

ADOBE READER	$5,000–$30,000
MAC OSX	$20,000–$50,000
ANDROID	$30,000–$60,000
FLASH OR JAVA BROWSER PLUG-INS	$40,000–$100,000
MICROSOFT WORD	$50,000–$100,000
WINDOWS	$60,000–$120,000
FIREFOX OR SAFARI	$60,000–$150,000
CHROME OR INTERNET EXPLORER	$80,000–$200,000
IOS	$100,000–$250,000

Source: https://www.troopers.de/wp-content/uploads/2013/11/TROOPERS14-What_Happens_In_Windows_7_Stays_In_Windows_7-Marion_Marschalek+Joseph_Moti.pdf , 2013

How to monetise a 0-day?

- White Market – go to “zero day initiative” - won’t get paid much

- Black Market
  - Use a broker – need to establish a relationship with one on the “underground”

Security Expectations

- Confidentiality
  - Maintained by encryption
- Integrity
- Authentication
- Availability
  - Resilience to DoS
  - Attacks that crash programs
  - Attacks that cause CPUs to run at 100%

Auditing and Black Box Testing

Auditing

- Auditing: analysing code (in source or in binary) to uncover Vulnerabilities

Situations where code auditing makes sense:

- In-house software audit (pre-release)
- In-house software audit (post-release)
- 3rd party product range comparison
- 3rd party evaluation
- 3rd party preliminary evaluation
- Independent Research

Lack of skills in this area

Auditing vs Black Box testing

- Auditing: analysing code (in source or in binary) to uncover Vulnerabilities
- Black Box Testing: a method of evaluating a software system by manipulating only its exposed interfaces
- Typically involves specifically crafted inputs to get the software to crash or expose data.
- Eg testing a HTTP server:
    - GET / AAAAAAAAAA.....AAAAAAA/ 1.0

Auditing vs Black Box testing

- Fuzzing: automated black-box testing
- Adv of Black Box : quick
- Disadv: you don’t see the code paths
- Example on next slide of code where a black box or fuzz test would likely miss a buffer overflow vuln.
- For best results, we need BOTH black box and code auditing

Code Auditing vs. Black Box Penetration TestingExample

```
struct keyval         {
            char * key;
            char * value;
}

int handle_query_string(char * query_string)
{
            struct keyval *qstring_values, *ent;
            char buf[1024];
            if(!query_string)
                        return 0;
            qstring_values = split_keyvalue_pairs(query_string);

            if(ent = find_entry(qstring_values, ”mode”))!= NULL)
            {
                        sprintf(buf, ”MODE=%s”,ent->value);
                        putenv(buf);
            }
            …
}
```

Vulnerability:
Programmer assumes that ent->value fits into buffer

ent->value is controlled by input

Classifying Vulnerabilities

Design Vulnerabilities
- SDLC phases 1,2 and 3

Implementation Vulnerabilities
- SDLC phases 4 and 5

Operational Vulnerabilities
- Flaws that arise in deploying and configuring software in a particular environment

Classification of Vulnerabilities

Design Vulnerabilities

Eg telnet is designed to allow for remote login. But it is DESIGNed to rely on unencrypted communication

IPv4 has ’idle host scan’ designed in

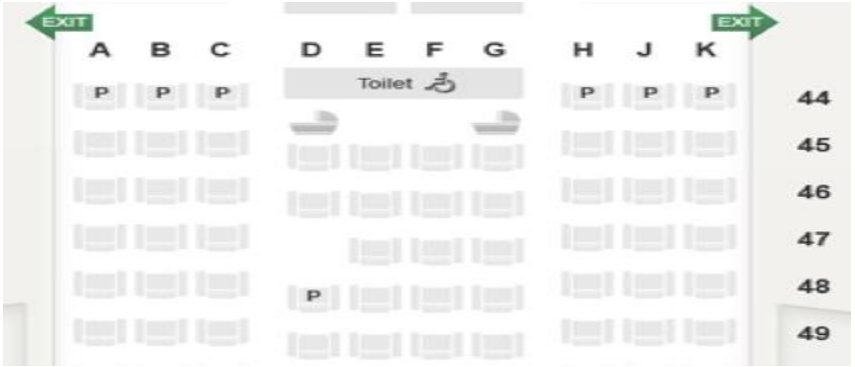
IPv6 has neighbour discovery DoS designed in

Design Vulnerabilities

Example: You are reviewing an application which is a seat booking system for an aeroplane.

- When the user clicks on a seat, it changes colour to orange, and the application temporarily reserves the seat for either 20 seconds or until the user clicks on a "reserve seat" button, which causes the seat to be assigned to the user, and the letter P is placed on the seat.
- While the seat is orange it is temporarily marked as booked on the backend of the application, and is unavailable to other users.
- If the user does not reserve the seat within the 20 seconds, the seat becomes free again, and it changes colour back to grey.

Design Vulnerabilities



Suppose a user wants a whole row to himself.

He books his seat, and then writes a script to continuously begin the booking of the other 2 seats in the row, every 20 seconds.

This is a logical or design vuln.

Implementation Vulnerabilities

Eg there may be a buffer overflow in a version of TELNET.

Some earlier versions of TELNET did not cleanse environment variables properly, allowing users to elevate privileges

Operational Vulnerabilities

Not in the source code

Can include

- Issues with the configuration of the software
- Issues with the configuration of supporting software
- Issues caused by processes surrounding the system
- Attacks on users of the system (social eng)

Operational Vulnerabilities

Eg TELNET in a system for automated trading

Every night, a set of weighting values need to be adjusted. Admin telnets in, and enters the new values. Depending on the environment, this is an operational vulnerability, due to sniffing of the credentials being possible

Grey Areas

Not always possible to categorise vulnerabilities as design, implementation or operational

Common Threads

Where and why vulnerabilities are most likely to surface

Common Threads

Input and Data Flow

Trust Relationships

Assumptions and Misplaced Trust

- Input
- Interfaces
- Environmental
- Exceptional Conditions

Input and Data Flow

Most Vulnerabilities triggered this way

Directly via user input

OR, via different sources and interfaces

- E.g. software that handles the scheduling of meetings, which generates a report at the end of every month, including a summary of each meeting. If this summary is > 1000 chars (say), we have a buffer overflow condition

To exploit this, the attacker creates a meeting with a description > 1000 chars, and then wait until month end to see if it worked

Input and Data Flow

The malicious data has to pass through several parts of the system, be stored in the Database etc.

The security auditor would have to trace the flow of data from entry to when it meets the application.

In a large system, this can be a complex task

Trust Relationships

Designers and developers often consider the interfaces between 2 components to be trustworthy

Sometimes trust is transitive e.g. app trusts component, which trusts the network, so the app indirectly trusts the network

Assumptions and Misplaced Trust

Developers can make incorrect assumptions about

- Validity and formatting of input
- Security of supporting programs
- Hostility of the environment
- Capabilities of attackers & users
- Behaviour of API calls

INPUT

Developer views input one way, while the attacker has a different view

If the code is asking for an address, the developer does not anticipate an address of size 1024 bytes

But the attacker thinks of all possibilities

Interfaces

Interfaces allow software components communicate with each other

A program component may be accessible via a network
Developer assumes that attacker cannot reach the component

Environmental Attacks

Developer makes assumptions about the underlying environment in which the software is running

Developer might not understand the security issues of the technology

Eg UNIX tmp race condition

- When a program needs to create a temporary file, it uses /tmp.
- On some flavours of Unix, if the filename exists as a symlink, the target of the symlink will be created; this can be exploited to create an arbitrary file on the system.
- The attacker can get the program to create its temporary file somewhere else on the system and with a different name
- This is an attack against the program's runtime environment

Exceptional Conditions

Occurs when the attacker causes an unexpected change in the program's normal control flow via external measures

Eg

- send a signal such as a SIGINT
- Consume system resources to induce a fail condition

Example: NFS daemon crashed if the network connection was closed at a specific time, necessitating manual intervention to restart the daemon

Software Vulnerabilities Fundamentals

Dr. David Stynes