

Software Development and Information Security:
an analysis, investigation and experiment into
what happens when security is treated as an add-
on during development.

Tom Neaves

100512587

Supervisor

Professor Kenny Paterson

Submitted as part of the requirements for the award of the MSc in
Information Security at Royal Holloway, University of London.

I declare that this assignment is all my own work and that I have
acknowledged all quotations from the published or unpublished works of
other people. I declare that I have also read the statements on plagiarism
in Section 1 of the Regulations Governing Examination and Assessment
Offences and in accordance with it I submit this project report as my own
work.

Signature:

Date:

Abstract

The ultimate problem in Information Security lies with Software Security. Developers are not being educated in how to properly implement Information Security into Software Development. This of course leads to vulnerabilities in programs; some minor, some major, but vulnerabilities none the less. The amount of vulnerabilities appearing in code can be reduced, again, with the use of education. It is also important to note that implementing Information Security into Software Development isn't just about focusing on the coding side of things alone. It is about considering all the various aspects of security (threats, risks, etc.) right from the start and involving them throughout the development of an application. This project investigates the possible reasons why a gap between Software Development and Information Security exists. We discuss and analyse various approaches to implementing Information Security into Software Development in the form of Security Development Lifecycles, Threat Modelling, Risk Analysis, etc. We then experiment in the form of designing and implementing a security application that will act as a case study; Bluetrak, (which will track Bluetooth devices), but designed purposely without considering security any more than usual. Bluetrak is then analysed constantly in terms of what we did compared to what we should have done in relation to developing secure applications. This is to demonstrate why treating security as an add-on really doesn't work. We then apply all the knowledge that we have learnt to Bluetrak in the form of a Security Assessment. However, we also highlight the ineffectiveness of the Security Assessment because security has only been considered later on in the development, rather than right from the start. Of which, the consequences of doing so are soon apparent.

The aims of this project are to demonstrate and prove why treating security as an add-on doesn't work (through the Bluetrak implementation and the late Security Assessment) and to educate on how exactly to go about implementing Information Security into Software Development properly.

Acknowledgements

First and foremost I would like to thank Professor Kenny Paterson for continually delivering his comments and feedback with a much needed slice of humour.

I would like to thank my mum, my dad and my sister for supporting me when I decided to prolong being a student by another year to do this MSc and for allowing me to pursue my dream of being a security consultant since I was 12 years old.

I would also like to thank my wonderful girlfriend, Danielle, for persevering with me when I'd be out the door at 5am and not back much before 8pm most days due to travelling to Royal Holloway. On that note I'd also like to thank Railtrack for only making me late twice, once of which was before an exam!

Lastly, I would like to thank all the wonderful and talented people (both students and lecturers) who I've met at Royal Holloway on this MSc who have in some way added value to both this project and my experience.

I dedicate this to Tangle (my black Labrador of 11 years) - who has bravely been fighting mouth cancer since he was first diagnosed with it back in April of this year. You served as my motivation for this project; you haven't given up, and neither have I.

For Tangle

Contents

<u>Chapter 1: Introduction</u>	<u>6</u>
<u>Chapter 2: Design and Development of Bluetrak</u>	<u>8</u>
<u>2.1: Introduction to Bluetrak</u>	<u>8</u>
<u>2.2: Design and Concept</u>	<u>10</u>
<u>2.3: Database Design</u>	<u>16</u>
<u>2.4: The Client Application</u>	<u>19</u>
<u>2.5: The Server Application</u>	<u>22</u>
<u>2.6: Device Discovery</u>	<u>29</u>
<u>2.7: Database Connectivity</u>	<u>30</u>
<u>2.8: Database Interactivity</u>	<u>31</u>
<u>Chapter 3: Software Development and Information Security</u>	<u>32</u>
<u>3.1: Bridging the Gap</u>	<u>32</u>
<u>3.2: Software Security Best Practices</u>	<u>34</u>
<u>3.3: Software Risk and Threat Analysis</u>	<u>40</u>
<u>3.3.1: STRIDE</u>	<u>41</u>
<u>3.3.2: DREAD</u>	<u>43</u>
<u>3.3.3: Responding to Threats</u>	<u>46</u>
<u>Chapter 4: Security Assessment</u>	<u>48</u>
<u>4.1: Security as an Add-on?</u>	<u>48</u>
<u>4.2: Bluetrak Threats and Risks</u>	<u>50</u>
<u>4.2.1: Spoofing Identity</u>	<u>50</u>
<u>4.2.2: Tampering with Data</u>	<u>52</u>
<u>4.2.3: Repudiation</u>	<u>56</u>
<u>4.2.4: Information Disclosure</u>	<u>57</u>
<u>4.2.5: Denial of Service</u>	<u>58</u>
<u>4.2.6: Elevation of Privilege</u>	<u>60</u>

4.3: Summary of Bluetrak Threats and Risks	61
4.4: Code Review	62
4.4.1: Code Analysis	64
4.4.2: Never Trust your Inputs!	67
Chapter 5: Conclusion	70
Bibliography and Appendix	72

List of Tables and Figures

2.1: Bluetrak in Action	10
2.2: Bluetrak Information Flow	11
2.3: The principles of a pico net	14
2.4: The 'CLIENT_LIST' table	16
2.5: The 'BT_LOG' table	17
2.6: The 'BT_LOG_ARCHIVE' table	18
2.7: Flow Chart Diagram - Client Application Process	21
2.8: Clients List Form	23
2.9: Add/Edit Client Form	23
2.10: Manual Reloading	24
2.11: Automatic Reloading	24
2.12: Filtering	25
2.13: Archive prompt	26
2.14: Switching between tables - log radio buttons	26
2.15: Printing of logs	27
2.16: Sorting by 'Surname' in alphabetical order	28
3.1: A Security Development Lifecycle (SDL)	34
3.2: Microsoft's Security Development Lifecycle (SDL)	37
4.1: Threats (from STRIDE) and their Risk Level (from DREAD)	61
4.2: Unexpected Input in Bluetrak Client	67
4.3: Error Message due to Unexpected Input in Bluetrak Client	68

Chapter 1

Introduction

The future of information security relies on software developers taking security into consideration when writing applications. It is not acceptable for an application to be classed as 'secure' by merely focusing just on the code alone. Theresa Lanowitz of Gartner Inc. states that "75 percent of hacks happen at the application". This is an important statistic that needs to be acted upon. Developers need to stop thinking just about functionality and consider the whole design and business logic behind their application with regards to security. In an essence, developers need to be taught how to write secure applications, and properly. This is happening, however, at an extremely slow rate. Developers are more focused on features and performance over security. This is because features and performance sells. Security, unfortunately, does not. Developers see implementing security in their applications as something that handicaps development and seen as an add-on that is only thought about after the product has been released. This is due to a number of factors, mostly deadlines and tight budgets, but more often than not, it falls down to education. Some reasons that developers give for failing to consider security are; "Security is boring", "Security is often seen as a functionality disablement - gets in the way", "Security is difficult to measure" and "Don't know how".¹ Bill Gates mentions another frightening statistic in his speech at the RSA Conference in 2005 which proves the point about education again (or the lack of it), "64 percent of developers are not confident in their ability to write secure applications"². The problem of insecure software applies to all products, even those aimed at the security market. Even when security is considered by the developer, it is done so at a much later stage in the development of an application. Therefore it isn't anywhere near as affective as if they had

¹ Source: Microsoft Tech-Ed 2005

² Source: Microsoft Developer Research

considered security right from the start. There are major consequences of only considering security at the end of development or not at all; in terms of time, cost and the amount of silly vulnerabilities that will still exist in the application. The cost of addressing security issues only increases as the software design lifecycle proceeds, so it seems logical to get things right straight from the start. The only way vulnerabilities in software are ever going to be reduced is by developers taking Information Security seriously when developing an application and implementing it properly throughout. If they can't, there are plenty of security people about who should be able to help them.

Within this project, an application aimed at the security market will be developed. It will be developed purposely with features and performance in mind over security. This will form the basis of our investigation into the gap between Software Development and Information Security. We will look at how to incorporate Information Security into Software Development properly. We will investigate what happens if one does not consider security in the development of an application at all (which we are doing with Bluetrak) - or at a much later stage (treating security as an add-on). We will discuss the implications of developing an application in this way. We will then attempt to apply all of what we have discussed, compared and learnt from how to effectively involve Information Security into Software Development (from various Security Development Lifecycles, Best Security Practices to Software Risk and Threat Analysis). Throughout these stages we refer to the development of our application with regards to the discussion; what did we do, and what should we have done? This is carried out from the perspectives of both the Security Consultant and the Software Developer to add value and to enable us to see their contradicting viewpoints. Finally, we will demonstrate why security as an add-on really doesn't work with a well overdue Security Assessment of our application. This will hopefully demonstrate the problems caused by considering security only at a much later stage and not throughout the entire design and development stages.

Chapter 2

Design and Development of Bluetrak

In this chapter, we discuss the design and development of Bluetrak. To be able to break (or hack) an application, one first needs to understand how it is designed and works - this chapter will serve as that purpose. We watch Bluetrak develop from the initial design and concept stage through to the implementation and everything else in between; the design of databases, system components, etc. It is important to note that during the design and development of Bluetrak, security has not been considered anymore than usual. No Security Development Lifecycle (SDL) has been followed (see Chapter 3 - Software Development and Information Security). No Threat Modelling or Risk Analysis has been conducted. Again, this has been done purposely to demonstrate how hard conducting a Security Assessment (Chapter 4) will become when things like "Abuse Cases" (discussed later) do not exist to work from. The implementation of Bluetrak serves as an entire case study which is touched on in following chapters.

2.1 Introduction to Bluetrak

Bluetrak is a piece of software created by myself that consists of a client part and a server counterpart. Bluetrak utilizes Bluetooth¹ (IEEE 802.15) technology to track, position and log Bluetooth enabled objects within a real world environment. Bluetooth² operates in the 2.4 GHz ISM band and is already embedded into most phones, PDAs, PC peripherals, etc. Bluetooth tags are small pocketsize transceivers. As with any other Bluetooth device, each tag has a unique ID. This ID can be used for locating and tracking the tag.

¹ <http://www.bluetooth.com/Bluetooth/Learn/Technology/>

² <https://www.bluetooth.org/spec/>

Bluetrak consists of nodes (the client application) scattered across and throughout the environment that act as listeners. All these 'clients' report back to a central database in real time (with additional details, location, timestamps, etc) to provide an audit log. Bluetrak also consists of a server counterpart - which interacts with the database to provide added functionality. For example, filtering, real time updates from all the listening nodes, creating relationships between employees and their relevant unique Bluetooth ID, etc.

Bluetrak is a security product that deals with physical security in the real world providing real time monitoring, tracking and auditing of people and/or assets in relation to locations in a real world environment. It is not an access control, although, with some modification it could be used as one.

Bluetrak's main goals are to provide identification in relation to tracking and accountability¹ in the event of a security incident occurring.

Bluetooth seems to always be in the media, but for all the wrong reasons. Bluetooth has had more than it's share of security problems, mainly due to the technology's design and implementation. However, people tend to overlook how Bluetooth can be used as a security aid as opposed to an obstacle in their way. Bluetooth has numerous benefits, of which are hardly ever mentioned by the media. Bluetrak will attempt to demonstrate these benefits.

¹ "Accountability - the property that ensures that the action of an entity may be traced uniquely to the entity." (ISO/IEC 7498-2)

2.2 Design and Concept

Figure 2.1: Bluetrak in Action

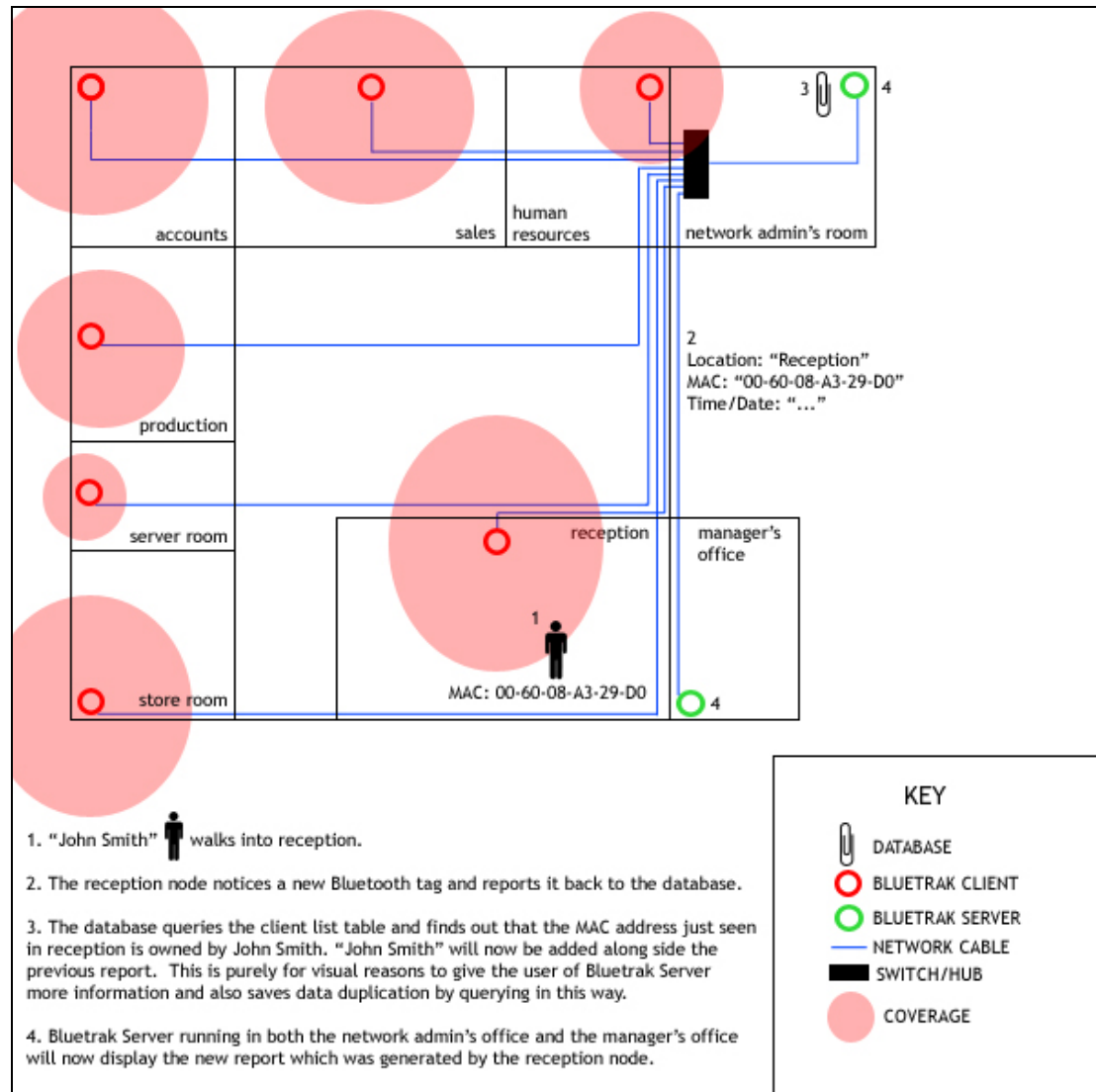
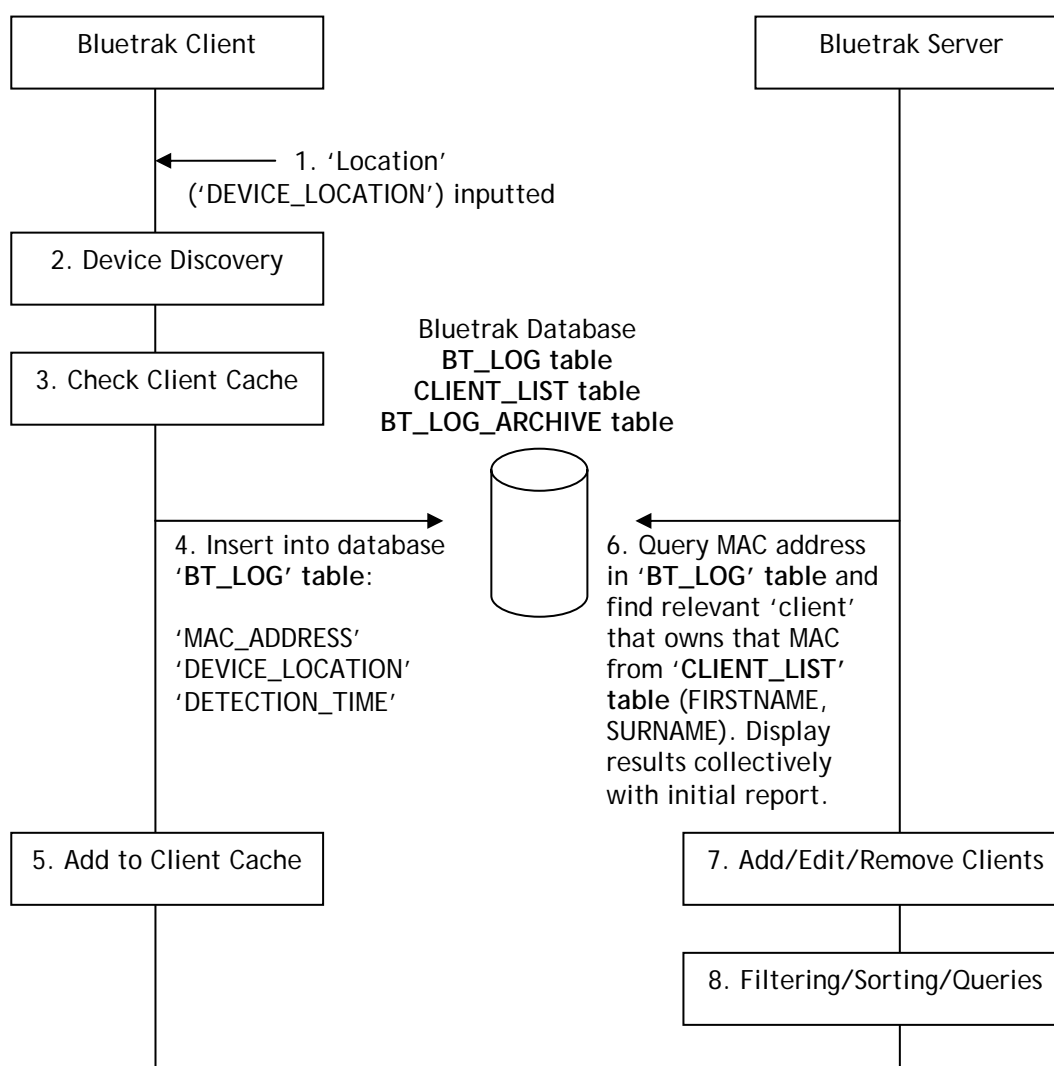


Figure 2.2: Bluetrak Information Flow



1. The 'Location' variable is inputted upon start-up of the client application. This uniquely identifies which area each node is listening in.
2. The client application searches for Bluetooth devices in its range.
3. The client application checks its own cache to see if the device is new or if it has been seen before (within maximum inactivity time - discussed in more detail later).
4. If the device found is new, add it into the database in the 'BT_LOG' table.

5. Add the newly found device into the client cache for the duration of the maximum activity time so multiple reports of this node are not continuously being added to the database.
6. Check the database for new reports (or records) and query the 'CLIENT_LIST' table to find who owns the relevant MAC address showing in each record (FIRSTNAME, SURNAME). Display collectively 'on the fly' in the same table with the initial report. This is extremely dynamic as someone's old Bluetooth tag could have been assigned a new owner - Bluetrak will reflect this in real time.
7. Ability to add/edit or remove client's details in the 'CLIENT_LIST' table (assign a new MAC address to a new client for example). Because of the dynamic queries on each record in step 6, the new results will show instantly in the main reports.
8. Additional filtering, sorting and queries on the main reports - by name, MAC address, time, location, etc. (discussed more later).

To summarise,

Client Application

Before the user starts scanning, they must first input a location value along with the scan interval and maximum inactivity time. Usually the last two are left at their defaults. Once the user has inputted a location value they may start scanning the environment for Bluetooth devices. Once one is found, it is then checked against its own cache to determine if it is a new device or has been discovered early. If it has been discovered before (within the maximum inactivity time), then it will let the user know of this fact but will not write to the database. If it is new, it will write to the database, alert the user of this action and also ignore it for the duration set in the maximum inactivity time (add device MAC address to its cache).

Server Application

The user uses this application to see all the live reports that are being sent from the clients. Additionally, they can interact with the database.

The whole concept of Bluetrak relies on static and mobile 'nodes' transmitting and receiving, so it is important that we are familiar with the concept of these first.

Bluetooth transmitters/receivers come in two classes; Class 1 - which has a transmission range of up to 100m (300ft), and a Class 2 - which has a range of only 10m (about 30ft). This is an important factor to consider during the design stage. The 'mobile' devices that are to be located will be Class 2 - they will transmit within a 10m radius. The justification for this falls down to power consumption. People will be wearing these Bluetooth devices (bodytags¹ and/or watchtags²) and therefore battery life is important. It is hardly practical if a battery needs to be recharged every hour or so. The devices provided by BlueLon³ are ideal for this. The 'static' devices (or nodes) that will be 'listening' out for these mobile devices will be in the form of Bluetooth USB dongles connected to computers. These will be of Class 1 as power consumption is not an area of worry as this will be provided by the computer it is connected to. That means the mobile devices now have a range radius of 100m too. They are not being used to transmit anything (that is not their use) - Bluetrak simply wants to be able to identify them (by their MAC address) and assign them to a specific location.

A mobile device (or mobile entity) represents a person, or an asset - whichever was assigned during the 'registration' stage (discussed later). What happens if the entity goes out of the 100m range? Simple, another node (or static device - a computer with a USB dongle) picks them up. Each computer with a USB dongle will act as a listener and will represent and cover a geographical area in the real world. This becomes problematic as each listener is now isolated from all the other nodes and are not nodes at all. For example, each will hold their own individual reports of devices that were seen throughout a specific time period. This creates a major problem in terms of integrity, not to mention efficiency and operational problems.

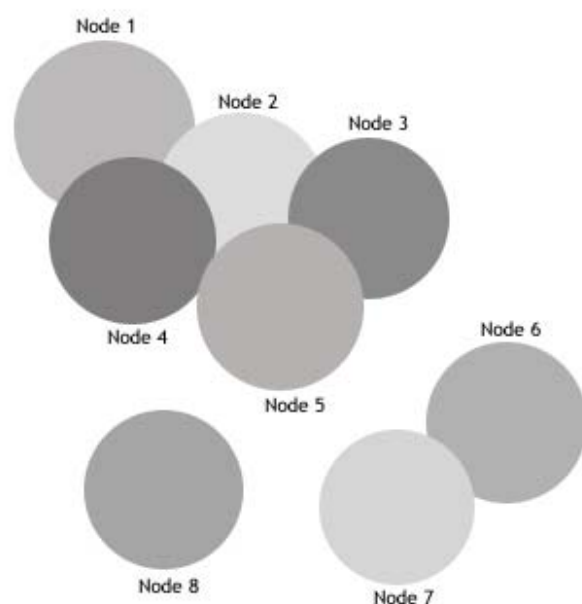
¹ BodyTag - http://www.bluelon.com/uploads/media/BodyTag_BT-002_ENU.pdf

² WatchTag - http://www.bluelon.com/uploads/media/WatchTag_WT-004_ENU.pdf

³ BlueLon - <http://www.bluelon.com>

These nodes need to work together collectively (a network) and report back to a central location if these are to be nodes at all. The ideal solution for this problem would be a central database. All nodes would report to this central database instead of logging locally. This would then create a PAN (personal area network) in a way - a pico net of Bluetooth nodes spread out geographically working together as one. This now means the range of the pico net is dramatically increased based on the number of nodes as the range is now combined. (See Figure 2.3 below for an explanation)

Figure 2.3: The principles of a pico net



Imagine Node 1 represented Office A, Node 2 = Office B, Node 3 = Office C and Node 8 = Reception. If a mobile Bluetooth device was within a 100m radius of any of the nodes, it would be picked up and reported back to a central database. It is important to emphasise the 'any' because the range is now combined and thus extended - they are working as a pico net. Therefore, even if the mobile device was in Reception (which is obviously way out of the combined radius of Nodes 1-3), it would be in the radius of Node 8 - which would report it to the central database.

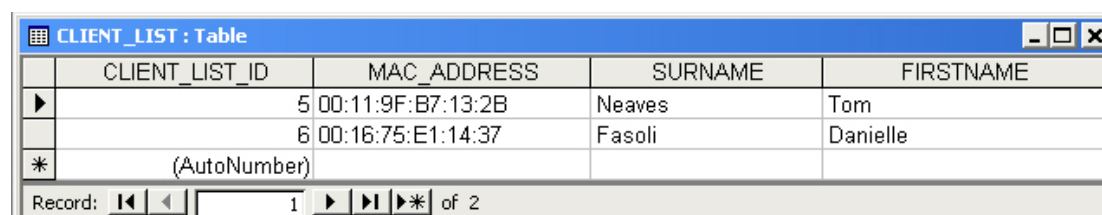
The diagram on the previous page illustrates the principles behind a pico net, however, also illustrates another point (or problem). What happens when a mobile device is within range of Node 1 and Node 2? Obviously and logically, both static devices will report the mobile device to the central database. The problem of this is soon apparent. The database will contain two records, one made by Node 1 stating the relevant mobile entity is in Office A, and another made by Node 2, stating that the mobile entity is in Office B. In mathematical terms, this is the subject of set theory; elements, unions, etc. and can be solved in this way. However, the simple solution would be to position static nodes (computers with Bluetooth USB dongles - 'listeners') in such a way that they do not overlap. In a perfect world, this solution would be perfect; however, this creates another problem. In a real world environment, offices are not 100m apart. A solution to this problem would either be to substitute a Class 1 USB dongle with a Class 2 (on the static node), thus reducing its range to 10m, or, by making one node cover an area in a more general sense. For example, splitting them up into larger blocks; "Office Block A", "Lower Offices", "Human Resources", etc. This of course reduces the ability to accurately pinpoint an entity's exact location but it is a sacrifice that will sometimes have to be made. By reducing a static node to a Class 2 USB device enables the pico net to still function in greater detail and would avoid the overlapping issue. This Class substitution technique should be used where possible over the 'generalising a geographical area' approach which would lose the detail of information that is provided which in turn affects the accuracy of tracking.

2.3 Database Design

The design of the database is extremely important. The database will be the 'backend' of the entire pico net in a way. Without it, the pico net is not a pico net at all; it is merely individual nodes reporting locally. The database allows for collaboration of all nodes to enable them to work as a team, reporting remotely to a central location; the database. Bluetrak's goals are to provide identification (in terms of real time tracking) and accountability (in terms of an audit). Therefore we bear these two goals in mind when approaching the database design.

Identification in this case is the Bluetooth device ID, which is also its MAC (media access control) address. A Bluetooth device ID (or MAC address) will represent a person or an asset (an entity) virtually in a database. These MAC addresses need to be somehow assigned (or linked) to a specific person (the person to whom it will represent in the database). This process will be referred to as 'registration'. This information will be held in the employee/client table. (see Figure 2.4 below)

Figure 2.4: The 'CLIENT_LIST' table



CLIENT_LIST_ID	MAC_ADDRESS	SURNAME	FIRSTNAME
5	00:11:9F:B7:13:2B	Neaves	Tom
6	00:16:75:E1:14:37	Fasoli	Danielle

* (AutoNumber)

Record: 1 of 2

CLIENT_LIST:

CLIENT_LIST_ID as integer, primary key, not NULL

MAC_ADDRESS as string, not NULL

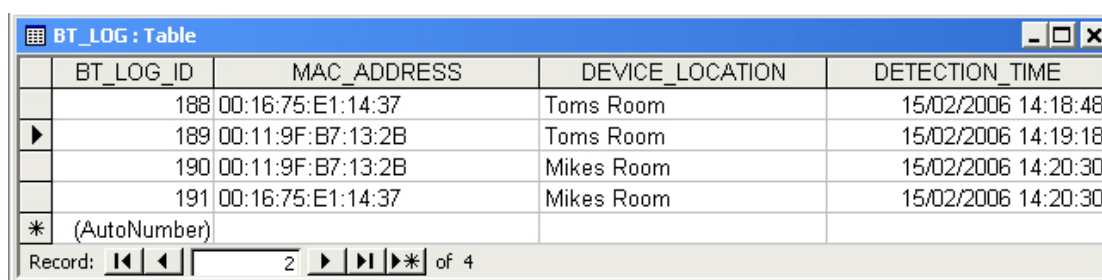
SURNAME as string, not NULL

FIRSTNAME as string, not NULL

Now a person is assigned to a tag (and vice versa) - each node on the pico net can now report meaningful information back to the central database. However, another table is required.

The whole concept of Bluetrak is the ability to carry out real time tracking of people/objects (assets) in a real world environment. Alternatively it can be used as an audit log after a security incident has occurred. To do this, each pico node needs to report specific information back that can be acted upon. We've identified WHO based on client to tag ID (MAC), now we want to identify WHO is WHERE, based on location. As mentioned earlier, each client application on a node will have a variable that needs to be entered when started up. This is the LOCATION variable. It is unique to each node and represents its physical location. This variable will be reported back to the central database each time a tag is discovered, along with MAC_ADDRESS of the tag, which in turn the server application queries the CLIENT_LIST table and gets FIRSTNAME and SURNAME relevant to that tag ID (MAC). Additionally, DETECTION_TIME will be appended onto the end of this report as without it, the information is again meaningless. BT_LOG_ID is simply an incremental auto number to distinguish between records.

Figure 2.5: The 'BT_LOG' table



BT_LOG_ID	MAC_ADDRESS	DEVICE_LOCATION	DETECTION_TIME
188	00:16:75:E1:14:37	Toms Room	15/02/2006 14:18:48
189	00:11:9F:B7:13:2B	Toms Room	15/02/2006 14:19:18
190	00:11:9F:B7:13:2B	Mikes Room	15/02/2006 14:20:30
191	00:16:75:E1:14:37	Mikes Room	15/02/2006 14:20:30

* (AutoNumber)

Record: 2 of 4

BT_LOG:

BT_LOG_ID as integer, primary key, not NULL

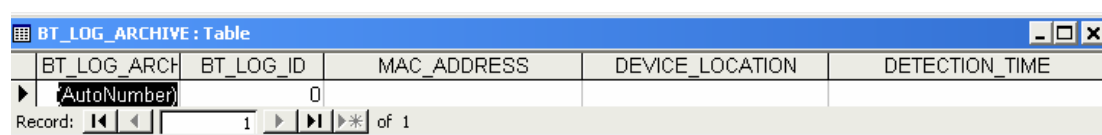
MAC_ADDRESS as string, not NULL

DEVICE_LOCATION as string, not NULL

DETECTION_TIME as timedate, not NULL

A further table will be added that will act as an archive log of the main BT_LOG table. This is because the BT_LOG table will need to be cleared out every so often as it will get extremely large in size with numerous records being added throughout a set time period. The decision when to archive will be left up to the security manager (and possibly when is defined in the security policy) and can be done via the Bluetrak Server application. The BT_LOG_ARCHIVE table looks exactly the same as the BT_LOG table (see Figure 2.6 below). All that will happen is that every record will be moved over if the decision to archive is taken and the BT_LOG table would then be empty.

Figure 2.6: The 'BT_LOG_ARCHIVE' table



The screenshot shows a window titled "BT_LOG_ARCHIVE : Table". The table has five columns: BT_LOG_ARCH, BT_LOG_ID, MAC_ADDRESS, DEVICE_LOCATION, and DETECTION_TIME. The BT_LOG_ARCH column is highlighted and contains the text "(AutoNumber)". The BT_LOG_ID column contains the value "0". The record number "1" is shown in the bottom left corner, and "of 1" is shown in the bottom right corner.

BT_LOG_ARCH	BT_LOG_ID	MAC_ADDRESS	DEVICE_LOCATION	DETECTION_TIME
(AutoNumber)	0			

BT_LOG_ARCHIVE:

BT_LOG_ARCHIVE as integer, primary key, not NULL

BT_LOG_ID as integer, not NULL

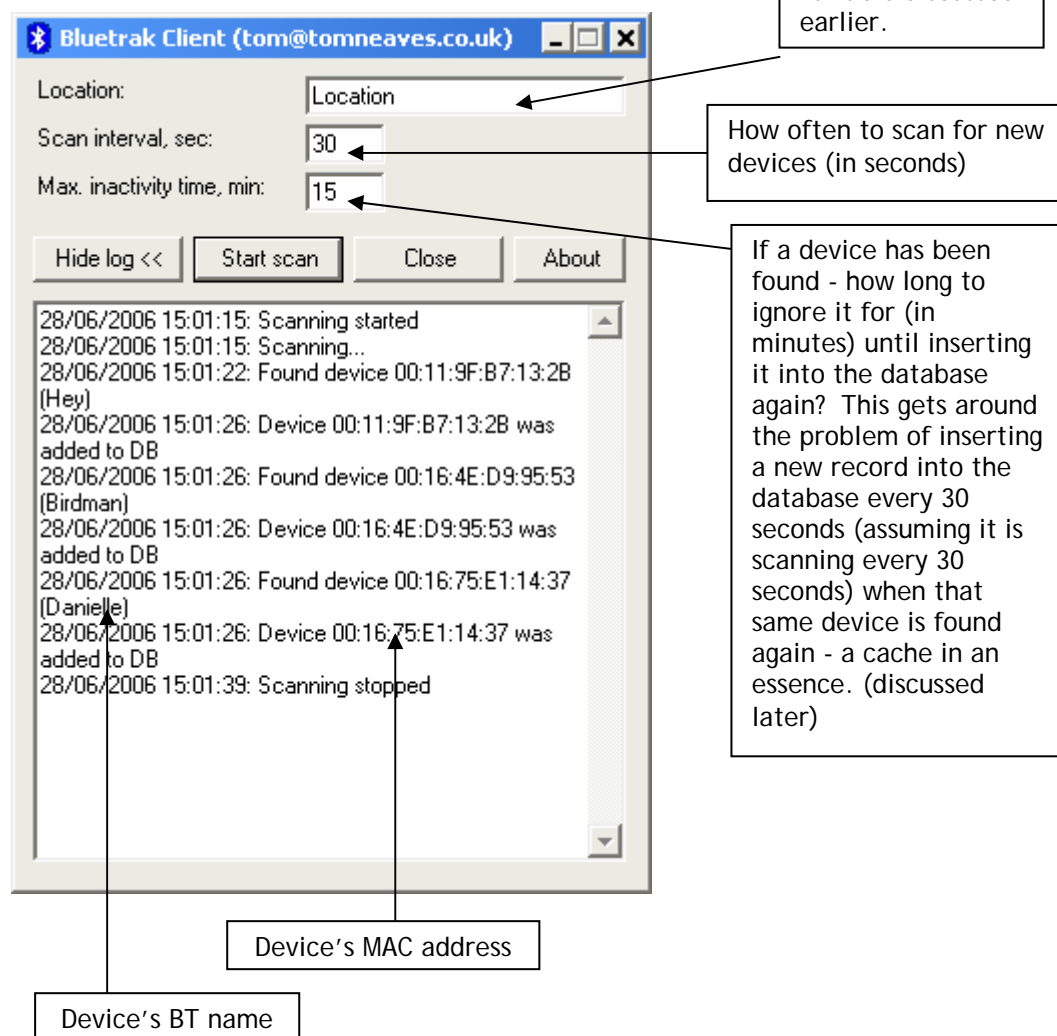
MAC_ADDRESS as string, not NULL

DEVICE_LOCATION as string, not NULL

DETECTION_TIME as timedata, not NULL

The BT_LOG_ARCHIVE table above is empty because nothing has yet been archived into it.

2.4 The Client Application



The client application represents a unique node (or listener) within the pico net of Bluetooth nodes making up the Bluetrak network. The node's job is to scan the geographical area (within its range) for new 'mobile' Bluetooth devices. The emphasis on 'mobile' Bluetooth devices is important - it will not pick up other 'static' nodes (the listeners) on the network as they are not broadcasting or transmitting anything. This is exactly what we want it to do; else each node would simply be reporting other static nodes back to the database in a continual loop.

Once a device is found, it will first report it on screen to the client application. It will then check its cache to check if the device has already been found (this depends on inactivity time set). If the device hasn't been

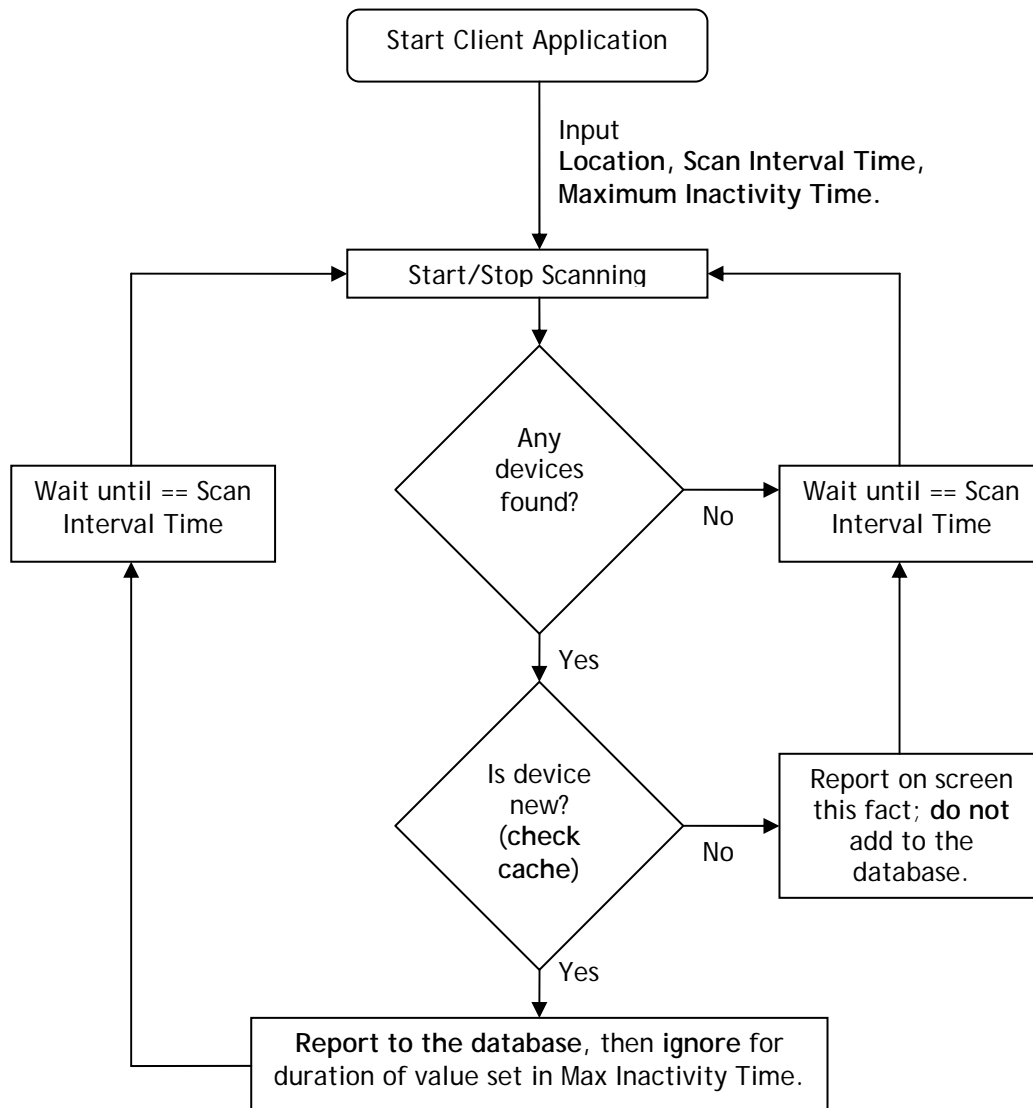
found before (or isn't within the max inactivity time) then the device is reported to the central database remotely (with timestamps, location, etc). If it has been found before (and is within the max inactivity time) then it will report on screen this fact and will not add it to the database. The maximum inactivity time field (or variable) is there to stop the client application continually reporting the discovery of the same device every 30 seconds to the database. For example, an employee may stay in a room for 5 minutes, or even 5 hours - does the database really need to be updated every 30 seconds with its location? The maximum inactivity time needs to reflect this based on the situation. The important factor here is that the inactivity time is unique to every node. For example, "Node A" has already discovered and reported "BT Device 1" to the database and is now ignoring it for 2 minutes. If "BT Device 1" then comes into the range of "Node B" (walks from Reception to Office A) then "Node B" will see it and report it to the database (as it's a new discovery for "Node B"). Therefore, the maximum inactivity value assumes approximately what is the acceptable level of time that device (person) will be in that environment (or room) for. It will ignore it for the duration based on this time, however, other nodes will pick up any movement and it will be reported back to the database with its new location. This is a little trick to get around the fact that the database will be filled up extremely quickly with reports of the same node every 30 seconds.

It is also important to note that the nodes will scan every 30 seconds for new devices (default set), which can be changed. The maximum inactivity time only affects devices that have been discovered before. Again, the maximum inactivity time needs to reflect the activities of the location the node is in. For example, Reception would have a low inactivity time as people are moving about (coming and going) all the time and we wish to know if they are still there. Offices would have a high inactivity time as the employee (the device on them) may remain in the same location for hours (inactive) - or we'd assume so. This only affects devices discovered previously unique to that node. If a device moves into the range of another

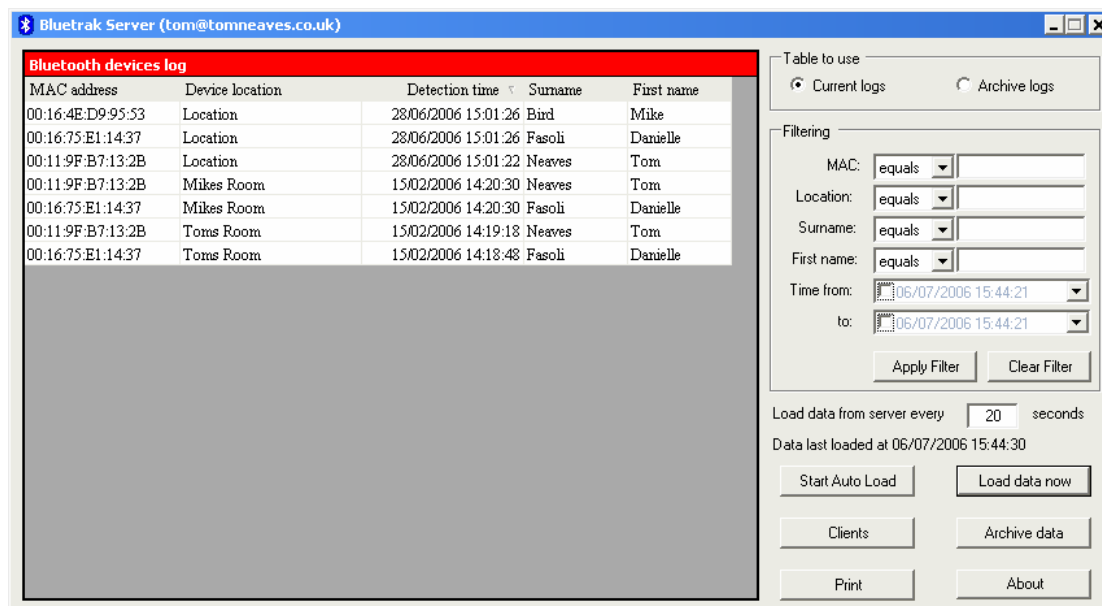
node (active), even if the previous node was ignoring it, it will be reported back to the central database by the new node, and so continues.

The flow chart diagram below will demonstrate this more clearly.

Figure 2.7: Flow Chart Diagram - Client Application Process



2.5 The Server Application



The server application interacts with the database much like the client application. However, it does so in a much more advanced way. The server application of Bluetrak gives the user the ability to manipulate the database in real time as they see fit. For example, custom SQL statements and queries can be created and issued to the database 'on the fly' in real time. Custom sorting can be carried out (filtering). Client (or employee) entities in the database can be added, edited or removed, etc. These are just a few examples of the powerful features available to the end user of the server application. All these features will be discussed in full in the following sub-sections.

Adding/Removing/Editing clients ('Registration')

Within the server application the user can access the client (or employee) table and edit it as they wish. This feature of the application will often be used only for the process of 'registration'. This process occurs when a new employee joins the company (or is issued with a Bluetooth tag). The employee needs to be linked with their tag on the system (as discussed earlier in the Bluetrak concept). This is carried out by entering the employee's details (First name and Surname) followed by the MAC address

of their Bluetooth tag device. Now that employee is identified by their unique MAC address in the database (see Figure 2.8 and 2.9 below).

Figure 2.8: Clients List Form



Figure 2.9: Add/Edit Client Form

MAC address:

Surname:

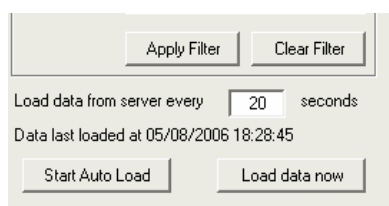
First name:

Automatic/Manual Database Reloading

The user has the ability to automatically reload/update the database after so many seconds, i.e. 20 seconds (which is the default). Or, the user can manually reload the database as they wish and disable automatic reloads (see Figure 2.10 on the next page). Concerning automatic reloading (see Figure 2.11 on the next page), the user can have the database being reloaded every 1 second making it real time. It is important to note that

this 'reload' time is independent of nodes (running the client application) reporting back to the database. They will still be writing to the database, however, it will not show up in the server application until the database is reloaded. This feature was added to give as much flexibility as possible in relation to varying network infrastructures and topologies. The bandwidth might not be available to reload every second, some users may find that real time updating is not something they want or require. For example, they may not be sitting there watching it all day and it would be a waste of network resources to do so. It is also important to note that the updating (or reloading) is merely a visual feature of the server application. This means that again client nodes will still report to the database even if the server application isn't running. The benefits of this are quite obvious as network and processor resources are not being wasted. The server application acts much like a remote desktop feature in an operating system - even if we are not connected to the machine remotely, the processing is still going on in the background. Some users will require Bluetrak to report back to them in real time, for which it is capable of doing so and a few clicks will enable this. The database will initially be loaded when the server application is first started.

Figure 2.10: Manual Reloading



User manually clicks 'Load data now' button.

Figure 2.11: Automatic Reloading

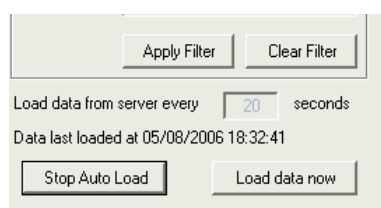
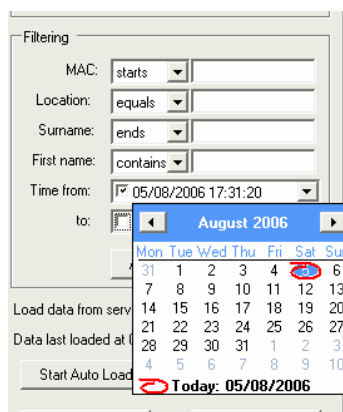


Figure 2.11: The user has inputted to reload automatically every 20 seconds, and then has clicked the 'Start Auto Load' button which has now turned into 'Stop Auto Load' and has disabled the seconds input textbox. The user can still manually reload if they wish while automatic reloading is selected. Or, they can click the 'Stop Auto Load' button to be able to edit the seconds variable or to stop automatic reloading of the database.

SQL and Filtering

Within the server application the user has the ability to filter the results on the main table (or data grid) to the left (the BT_LOG table). This can be done by 'MAC' address, 'Location', 'Surname', 'First name', 'Time from' and 'to'. The SQL can also be manipulated within these filters by use of combo boxes (or drop down menus) giving the user options such as; string 'starts' with, 'equals', 'ends' or 'contains'. This is relatively easy to use (see Figure 2.12 below). For example, if we want to filter the database to show only John Smith's whereabouts, we can type in John Smith into the 'Surname' and 'First name fields' and leave it as 'contains', or, if we know his MAC, type that in and then click on the 'Apply Filter' button. All new reports from client nodes will still write to the database, however, they will not show up currently in the server application table to the left as this filter is in place for "John Smith." To get things back to normal all that is needed is to click the 'Clear Filter' button. This filtering also works for the archive table, which will be discussed next.

Figure 2.12: Filtering

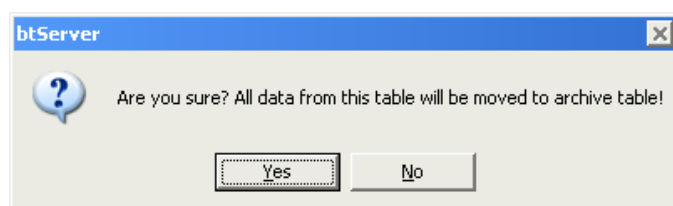


Archiving

As mentioned earlier (mainly in the database design section), the server application has the ability to archive logs into an archive table, so that the main table is clear and doesn't get too big.

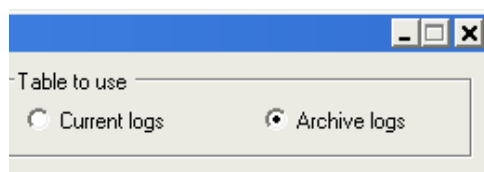
This is done by clicking on the 'Archive data' button to which the user will be prompted with a message box making sure this is what the user wanted to do. If it wasn't, they can of course cancel by selecting 'No'. However, if it is what they wanted to do, 'Yes' is selected (see Figure 2.13 below). All the records in the main table (BT_LOG - the one showing on the left of the server application window) will now be moved to the archive table (BT_LOG_ARCHIVE).

Figure 2.13: Archive prompt



The user can still view the archive table if they wish by the use of radio buttons in the server application which allow the user to switch between 'Current logs' and 'Archive logs.' See Figure 2.14 below for an example of these radio buttons. The table to the left of the application will switch to the relevant table according to which radio button is selected.

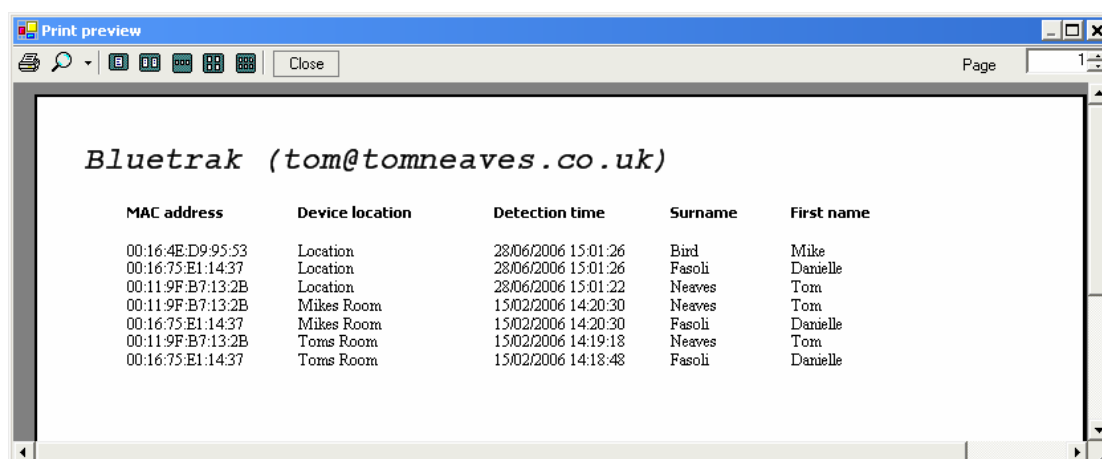
Figure 2.14: Switching between tables - log radio buttons



Printing

The server application of Bluetrak enables the user to print the contents of the 'current log' (BT_LOG) or 'archive log' (BT_LOG_ARCHIVE) at any time depending on what radio button is selected (see section before). Additionally, this feature is dependent on the filtering. For example, if a filter for "John Smith" is active, the only records that will be shown on screen in the table are for "John Smith" only. If the 'Print' button is then selected, a print preview of only John Smith's records will be shown. If no filters are active, all entries will be shown ready for printing (see Figure 2.15 below).

Figure 2.15: Printing of logs



Bluetrak (tom@tomneaves.co.uk)

MAC address	Device location	Detection time	Surname	First name
00:16:4E:D9:95:53	Location	28/06/2006 15:01:26	Bird	Mike
00:16:75:E1:14:37	Location	28/06/2006 15:01:26	Fasoli	Danielle
00:11:9F:B7:13:2B	Location	28/06/2006 15:01:22	Neaves	Tom
00:11:9F:B7:13:2B	Mikes Room	15/02/2006 14:20:30	Neaves	Tom
00:16:75:E1:14:37	Mikes Room	15/02/2006 14:20:30	Fasoli	Danielle
00:11:9F:B7:13:2B	Toms Room	15/02/2006 14:19:18	Neaves	Tom
00:16:75:E1:14:37	Toms Room	15/02/2006 14:18:48	Fasoli	Danielle

Sorting

Sorting is much like filtering, however, it requires no SQL queries to be issued unlike filtering. Sorting takes place in real time within the table on view in the server application. This can be carried out by clicking on a column title which has a grey background in the cell. An arrow will then change from up to down or vice versa. This resembles sorting by ascending/descending order. It is possible to do this on any of the columns. For example, the user may wish to sort by 'Surname' but in alphabetical order. The user can do this by clicking on the 'Surname' column title so that the arrow is pointing up. The table will then be re-sorted and displayed by 'Surname' in alphabetical order. See Figure 2.16 below for an

example of this. To sort in reverse, the user must click again (the arrow will then point down).

Figure 2.16: Sorting by 'Surname' in alphabetical order

MAC address	Device location	Detection time	Surname	First name
00:16:4E:D9:95:53	Location	28/06/2006 15:01:26	Bird	Mike
00:16:75:E1:14:37	Toms Room	15/02/2006 14:18:48	Fasoli	Danielle
00:16:75:E1:14:37	Mikes Room	15/02/2006 14:20:30	Fasoli	Danielle
00:16:75:E1:14:37	Location	28/06/2006 15:01:26	Fasoli	
00:11:9F:B7:13:2B	Toms Room	15/02/2006 14:19:18	Neaves	Tom

Arrow is pointing up

User clicks here (Column Title)

Sorting can also be carried out (just like filtering) on the archive table in exactly the same way.

2.6 Device Discovery

Device discovery was very problematic initially. The drivers that came with the Bluetooth USB dongle that were being used installed the WIDCOMM drivers and had its own Bluetooth stack. However, hardly any documentation was available for these drivers. Also, they implemented their own custom Bluetooth stack and any kind of SDK (software development kit) was costly. After some research¹ it was discovered that Microsoft's Bluetooth stack that comes with Microsoft Windows XP Service Pack 2 would be perfect. Lots of documentation² was about, mostly via MSDN (Microsoft Developer Network). Uninstalling the WIDCOMM drivers and getting the MS Bluetooth stack installed was again problematic, however successful in the end.

*"To facilitate the discovery of Bluetooth devices and services, Windows maps the Bluetooth Service Discovery Protocol (SDP) onto the Windows Sockets namespace interfaces"*³

The MS Bluetooth stack did not provide any .NET compatibility therefore a .DLL (dynamic link library) had to be created in C++ which would communicate with the Microsoft Bluetooth stack via the Winsock API (application program interface). This uses sockets (as mentioned above as to allow SDP mapping). Winsock 2.2 was used.

The client application (coded in .NET C#) then uses this .DLL (btInfo.dll) to talk to the Bluetooth USB dongle and discover new devices. The server application does not use this .DLL as it has no use for it - it does not discover devices. The source code to btInfo.dll can be found in the appendix.

¹ "Bluetooth Stacks and Components" <http://www.softwaregreenhouse.com/>

² "Bluetooth for Programmers" - Huang and Rudolph - <http://people.csail.mit.edu/rudolph/>

³ "Discovering Bluetooth Devices and Services" - Microsoft Developer Network:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/bluetooth/bluetooth/discovering_bluetooth_devices_and_services.asp

2.7 Database Connectivity

The client connects to the database by parameters set in the btClient.exe.config XML file which resides in the same directory as the btClient.exe file.

btClient.exe.config:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
<appSettings>
<add key="DBConnectionString" value="Provider=Microsoft.Jet.OLEDB.4.0;Data
Source=C:\Program Files\Bluetrak\bluetrak.mdb;"/>
</appSettings>
</configuration>
```

The server application connects in exactly the same way and its contents are the same. The only difference being is the XML file containing the parameters is called btServer.exe.config and resides in the same directory as the btServer.exe file.

On start-up of both the client and server applications, these files will be parsed and the value inside "Source=" will be used as the target database to connect to. If this is not found an error will be displayed alerting of this fact. Chances are the database will be on a remote computer to the client applications. Therefore Microsoft Windows sharing will have to be used and "\\servername\shared_directory\database.mdb" entered within this "Source=" value on client nodes. Additionally, the server application may be connecting remotely; the same is true and still applies. This will be discussed in more detail later. Microsoft's Jet 4.0 (joint engine technology) Engine will be used via OLE DB (API) to connect to the database. OLE DB is a set of interfaces implemented using COM (component object model) instead of the out dated ODBC (open database connectivity). It is appreciated that MSJET 4.0 is extremely old and that Microsoft have since dropped it as part of their MDAC (Microsoft data access components) but it is more than sufficient for the needs of Bluetrak.

2.8 Database Interactivity

The client and server application both need to access the database for the whole concept of Bluetrak to work. Therefore it made sense to create another .DLL which both applications would share to do this. This would be the job of the btDAL.dll file (DAL relating to database access linker). The client application will simply be writing to the database. It will never carry out any queries, unlike the server application. The server application does the opposite to the client application; it mostly reads the database and carries out queries. The only time the server application needs to write to the database is when clients are being added/removed/edited from the CLIENT_LIST table, or, when archiving takes place - moving records from the main BT_LOG table to the BT_LOG_ARCHIVE table.

btDAL.dll contains pre-defined SQL statements for use by both the client and server application. Once the client application finds a new Bluetooth device (and after it has checked it's cache to check that it is actually new) it will use btDAL.dll to enable it to access and write to the database. It will use the SQL statement below to insert the strings within the client application into the fields in the database.

```
string sqlStr = "INSERT INTO " + ds.BT_LOG.TableName + "(" +  
ds.BT_LOG.MAC_ADDRESSColumn.ColumnName + ", " +  
ds.BT_LOG.DEVICE_LOCATIONColumn.ColumnName + ", " +  
ds.BT_LOG.DETECTION_TIMEColumn.ColumnName + ") VALUES ( " +  
"" + macAddress + ", " + "" + location + ", " + "" + detectTime.ToString("G") + "" + ");
```

As stated before, the server application also uses this .DLL but in a different way. It does use some pre-defined SQL statements for functions such as deleting a client (much like above). However, the server application mainly accesses this .DLL for the datasets contained within. These datasets are accompanied by XML schema which tells the application exactly what can and can't be done with them. These datasets allow the data grid to be filled with the contents of the actual database. btDAL.dll can be thought of as a 'go between' (or the middle man) for the database contents to be filled in the data grid and vice versa. btDAL.dll is acting as an API in a sense. Unlike btInfo.dll, btDAL.dll is used by both applications and coded in C#.

Chapter 3

Software Development and Information Security

In this chapter, we investigate why a gap exists between Software Development and Information Security. We explore what can be done about closing this gap in the form of trying to implement various aspects of Information Security into Software Development. We touch upon Security Best Practices; discussing and comparing various Security Development Lifecycles (SDL) from the likes of; McGraw and van Wyk, Howard and LeBlanc and Microsoft. Software Risk and Threat Analysis is then discussed, using both STRIDE and Microsoft's DREAD methodologies for Threat Modelling and Risk Analysis. Microsoft's new "Security Bulletin Severity Rating System" for categorising threats and ranking risks is then discussed, evaluated in terms of it's effectiveness with case studies and compared to DREAD. This chapter serves as a 'how-to' in relation to developing an application with security in mind right from the design stage, something which has purposely been left out during the development of Bluetrak (Chapter 2). This chapter also serves as preparation material for the testing that will be carried out in the following chapter - (Chapter 4, Security Assessment). This chapter is, in an essence, how we should have developed Bluetrak, but didn't. This chapter will prove why following various aspects relating to security during the design and development stage are so important and what happens when we don't.

3.1 Bridging the Gap

Software developers are not security consultants and security consultants are not software developers (well a few are, but not many), according to McGraw and van Wyk [GM05]. They state there is a gap between software development and information security. One only has to look at the design and development of Bluetrak to see why. At no point was security really covered or touched, even though the application is aimed at the security

market and aims to provide identification and accountability. This is probably due to the developer trying to tweak everything to work that he has to ignore security - because there's too much else to worry about. The problems experienced in "Device Discovery" (Chapter 2) prove this theory; things had to be tweaked extensively to enable them to work together. The main priority for developers when creating an application is making the application work. As mentioned in the introduction, developers are usually working on tight budgets, managing multiple projects at once. They simply cannot afford for an application not to work as outlined in the brief.

What McGraw and van Wyk are trying to get across is that somehow security needs to be mentioned in that initial brief; even before the design stage is touched. The only way this can be done is with education. Educating the developers in security, or, by having the security consultants collaborate with the developers when developing an application - a secure application. This is slowly happening from both sides of the fence. The MSc (Masters) in Information Security at Royal Holloway, University of London now has a "Software Security" module. Numerous books have been published on "Writing Secure Code" and even Bill Gates of Microsoft has announced that "security would be their number one priority."¹ The gap between software development and information security is not a new issue at all. It has existed since the first hackers were exploiting vulnerabilities in computer programs back in the 1980's. However, a little more attention is being paid to the subject as computers are increasingly more a part of our lives than they were back then. Thus, if an important computer system goes wrong these days, the probability of it affecting our everyday lives is extremely high.

However, having stated that more attention on the subject of "software security" is taking place, I feel a problem still exists. Too much is being focused on the actual code of an application and not enough common sense

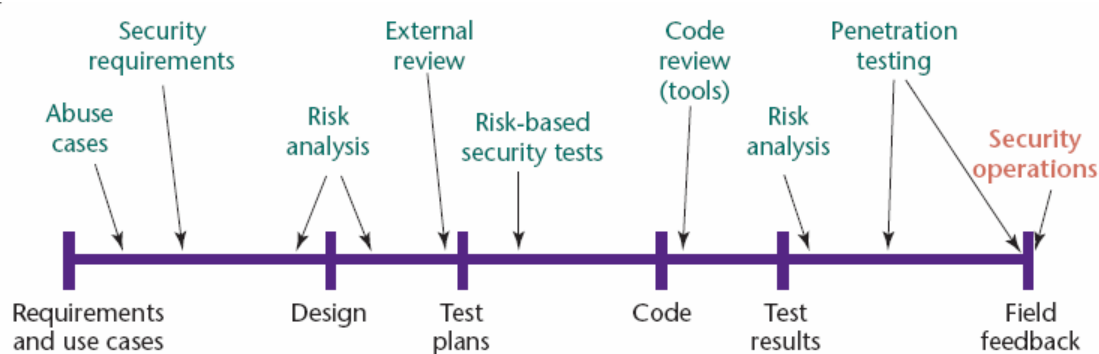
¹ "Gates: First job is secure software" - Seattle Times (17th January 2002)
http://seattletimes.nwsourc.com/html/business/technology/134392151_microsoft170.html

and logic is being applied. Not enough people are using the ability to look at an application from different perspectives and viewpoints. Standing back from an application and thinking 'outside the box' is required, thinking about both technical issues and business logic at the same time.

3.2 Software Security Best Practices

McGraw [GM06] introduces the concept of software security best practices or "touch points." - enabling a way to incorporate security into any software development. It looks very much like a normal software design lifecycle, however, contains new and important security subsections between the different stages, creating the "Security Development Lifecycle".

Figure 3.1: A Security Development Lifecycle (SDL)¹



Although this concept is not new (Howard and LeBlanc [MH02] originally came up with this idea, as did Microsoft), it does attempt to close the gap between software development and information security with the use of education.

Abuse cases focus on the developer considering deliberate misuse of the application, different scenario's; buffer overflows, malicious data, etc. and how the application responds to these.

¹ Figure taken from McGraw and van Wyk [GM05]

Security requirements is the stage at which the developer notes down what processes and controls need to be put in place to prevent (or try to reduce) the possibility of these abuse cases happening.

Risk analysis is the process of assessing the impact as a result of successful attacks outlined in the abuse cases. This subsection is important as it is divided up into two further sections; business risk analysis and architectural risk analysis. Both are very different in their approaches but have the same end result in mind. Business risk analysis looks at how successful attacks against the application could (and would) damage the business, in terms of liability, lost productivity and additionally indirect costs such as reputation and brand damage. Architectural risk analysis looks at the application in a more technical way. Detailed attention is paid to the proposed design with regards to security. Research is undertaken to find out the difficulty of possible attacks outlined in the abuse cases (and well as individually applied to each subsection of the design) along with what current exploits are published which are relevant to the application. McGraw and van Wyk [GM05] state that "50 percent of all security defects are architectural in nature."

Test planning is also split up into two further sections; security functionality testing and risk-driven testing. Security functionality testing is different to normal testing; it focuses on all the security elements in the application. For example, encryption, user identification, logging, confidentiality, authentication, etc. These will highlight the effectiveness (or ineffectiveness) of such security features which have been implemented. Risk-driven testing is the process of documenting findings from the architectural risk analysis in much greater detail which include how to actually carry out such an attack. Each of them is then individually given a priority (or risk level) based on their risk.

Code review occurs at the implementation stage but is very different to a normal code review. The code review is carried out solely with security in

mind. That is not only what makes it different from a normal code review. The factor that makes it different is that this code review focuses on finding bugs that were introduced during coding, e.g. buffer overflows. This is an issue that is sometimes overlooked because functionality is the main priority.

Penetration testing is applied to the application which focuses on human and procedural failures made during its configuration and deployment. This is based on the initial identified risks and its aim is to attain the level of exploitability of each.

Deployment and operations is part of the security operations stage in the field feedback section. This deals with the configuration and customisation of the application with security in mind. A security control which has been put in place to mitigate a discovered risk may not work if the application is not properly configured, rendering it useless and at risk. Close attention needs to be paid attention during this stage because the results could be dangerous and if done incorrectly could void all the hard done work before.

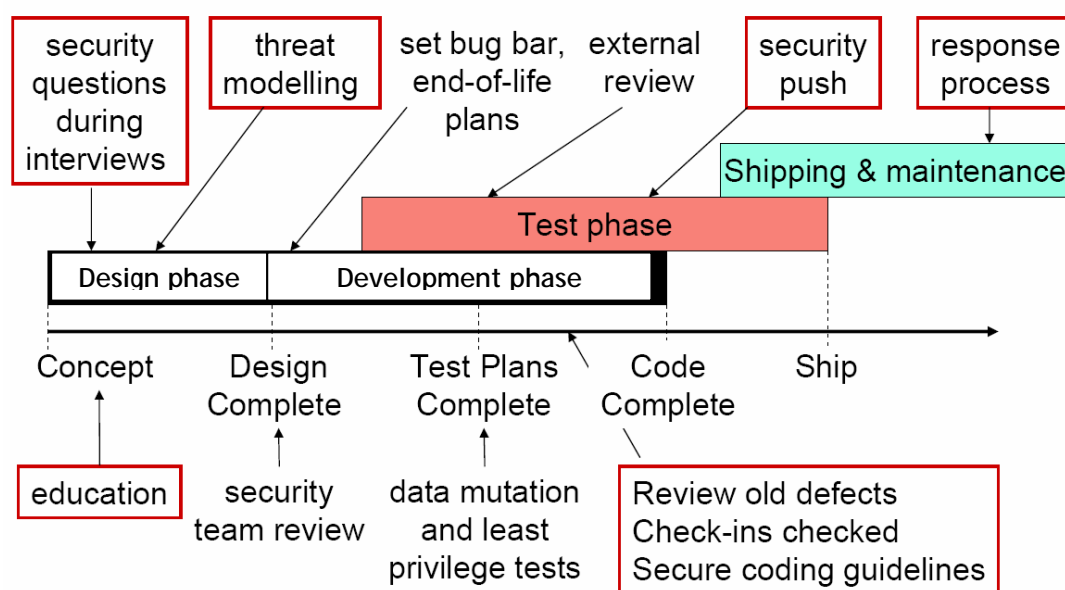
Microsoft's Security Development Lifecycle (SDL) looks very similar to the original SDL concepts by McGraw, Howard and LeBlanc, although has more emphasis on communication. Microsoft refers to these principles as "SD³+C" - "Secure by Design", "Secure by Default", "Secure in Deployment", and "Communications". These refer to:

- **Secure by Design:** the software should be architected, designed, and implemented so as to protect itself and the information it processes, and to resist attacks.
- **Secure by Default:** in the real world, software will not achieve perfect security, so designers should assume that security flaws would be present. To minimize the harm that occurs when attackers target these remaining flaws, software's default state should promote security. For example, software should run with the least necessary

privilege, and services and features that are not widely needed should be disabled by default or accessible only to a small population of users.

- Secure in Deployment: Tools and guidance should accompany software to help end users and/or administrators use it securely. Additionally, updates should be easy to deploy.
- Communications: software developers should be prepared for the discovery of product vulnerabilities and should communicate openly and responsibly with end users and/or administrators to help them take protective action (such as patching or deploying workarounds).

Figure 3.2: Microsoft's Security Development Lifecycle (SDL)¹



Microsoft has put more emphasis on getting rid of bugs during the design and development stage and even more focus on dealing with bugs and security incidents in the response process. Their SDL is already producing great results according to Jesper Johansson, Senior Security Strategist at Microsoft

¹ <http://msdn.microsoft.com/security/sdl>

in his keynote speak in 2005¹. According to Forrester², “Less total and high severity vulnerabilities existed in Microsoft operating systems in 2004 than in any other operating system. Faster fixes for publicly disclosed issues, faster than any other vendor - beating the likes of RedHat, Debian, MandrakeSoft and Suse - (25 days of total risk).” Johansson also stated that the SDL has had an impact on the design stage with security now in mind - their Microsoft Office product has had less security bugs reported after release than ever since their SDL has been implemented.

Concepts like these allow developers to incorporate security into the design and development of an application, whereas before it was seen as software development and information security were two different things.

Security needs to be at the fore front of application development and to be considered at every stage, not just seen as an add-on. One could argue that penetration testing carries an add-on “cow boy” like persona to it in relation to a security analysis. This has some truth to it, but only if that is the only time security has ever been considered in the application. Penetration testing simply checks that the controls and processes in place to mitigate (or reduce) risks are working correctly. If the developers have followed a security concept approach to development much like the one discussed before then these risks would have been identified already.

However, if security has not been a part of the development at all, then penetration testing will not be as effective and will be carried out blindly in a way. No scope would exist as to allow a penetration test to work from like abuse cases, scenarios, potential threats, vulnerabilities and risks, etc. along with any initial security requirements existing.

¹ <https://www.sicher-im-netz.de/content/sicherheit/ihre/software/KeynoteJohansson.pdf>

² Source: “Is Windows More Secure Than Linux?” Forrester, March 2004.

“Traditionally, security analysis has been applied at the network system level, after release, using tiger team approaches. After a successful tiger team penetration, specific system vulnerabilities are patched.” - McGraw [GM98]

The above quote is very much true of penetration testing and is very much effective. But again, is only effective if security has been considered throughout the design and development of the application. Else the penetration tester will have to start right from the beginning which will be an enormous job. That includes analysing code and the actual design and identifying security issues. This is not a lot of fun for the developer as he/she has to then go back and fix or redesign the whole application again in some cases, when they could have started that way initially and saved time and money. Additionally, the most embarrassing and ultimately dangerous thing about this is that the application has been ‘signed off’ and is in everyday use by customers. What happens if the penetration test has shown that the application is vulnerable to memory leaks? Vendors can’t simply just fix it and the job is done. They have to inform all customers (especially if that application is a financial related one - some countries have made it law that it is illegal to withhold this information). They have to get patches out, and quick. This is extremely bad and embarrassing for the company and the product’s reputation and could be devastating for their profits. This is covered in the business risk analysis (in the security development life cycle) and could have easily been avoided if it was followed. This is exactly why security needs to be involved right from the design stage right up until the end of that product’s life (and even beyond in some cases of providing legacy support).

3.3 Software Risk and Threat Analysis

Before we continue, it is important that we are familiar with the following definitions:

- **Vulnerabilities** are weaknesses of a system that could be accidentally or intentionally exploited to damage assets.
- **Threats** are actions by adversaries who try to exploit vulnerabilities to damage assets.
- **Risk** is the possibility that some incident or attack can cause damage to an organisation (and their assets).

To assess the risk an attack poses we have to evaluate the amount of damage being done and the likelihood for the attack to occur.

To fully be able to understand the threats that are posed to an application we must carry out a threat modelling process. This extends issues outlined in the initial "Abuse Cases" in the analysis stage. Threat models put security risks into a set of models which allow (and help) the development team to be able to design and create the proper security controls to put into place to reduce these risks (to be carried out in the business and architectural risk analysis stage). Threat modelling is a way of taking the threats that were defined at an analysis level in the "Abuse Cases" and apply real scenarios to them to discover their impact. It allows the development team to understand where the threats come from, how they may be carried out, in what environment are these attacks possible and the possible impacts. It is important to note that nothing practical occurs during this stage, it is the job of the "Testing" stage to extend the research found here.

Threat modelling consists of threat *identification* and *classification*. If a security development life cycle such as McGraw's [GM06] "security touch points" was followed, then an "Abuse Cases" section would exist. This makes the job of threat modelling far easier as there is something to work

off of. However, if security had never been considered in the analysis stage, then no section like this would exist and the job is going to be far harder. The first job of threat modelling is to identify what threats face the application. Each item in the “Abuse Cases” will be converted to one or more threat models. Of course not all threats will exist in the “Abuse Cases” section but a good majority will. That is why the process of threat modelling may have to be revisited later on in the development as new threats will be discovered and emerge. Threat modelling helps to find bugs, but most importantly design bugs that would not likely be found when looking at the individual components of the application.

Howard and LeBlanc [MH02] recommend a good way to start the process of threat modelling is to decompose the application and to identify key components and security boundaries. Howard and LeBlanc suggest that a better way of doing this is to look at high level data flow diagrams of the data flow between different components, rather than UML (unified modelling language) diagrams which are the old style and rather complicated. From looking at the data flow diagrams of each component with security in mind we are able to identify potential threats to the application (that may or may not have been initially identified in the “Abuse Cases”). This forms the identification stage of threat analysis. Now we need a way of classifying (or categorising) these identified threats.

3.3.1 STRIDE

STRIDE is a classification system that categorises threats as *Spoofing*, *Tampering*, *Repudiation*, *Information Disclosure*, *Denial of Service*, and *Elevation*. By using a threat classification system such as STRIDE gives an advantage over traditional software development methods as it allows us to see the application from an attacker’s point of view. STRIDE allows a structured way to categorize the relevant threats to the application based on the goals of an attacker.

Spoofing identity is where the attacker poses as another entity (e.g. user or server). For example, if we apply this to Bluetrak it could be the attacker pretending he/she is a client node, or, by spoofing a MAC address of a Bluetooth tag pretending he/she is another person. This is also referred to as impersonation, masquerading or identity theft.

Tampering with data is when an attacker makes unauthorised or malicious modification to data. In relation to Bluetrak, this could be modifying database records or capturing packets being sent to Bluetrak Server, modifying them and then replaying them. This is also referred to as (data) integrity.

Repudiation is when a user (wrongly) denies having performed an action, and there is insufficient evidence to prove the contrary. When relating to Bluetrak this could be a person denying they were in a specific area when the database has records that say they were. This is also referred to as accountability.

Information Disclosure is when information is revealed to an unauthorised entity. This could be an unauthorised employee being able to read the reports generated from all the Bluetrak Clients to Bluetrak Server in the database. This is also referred to as confidentiality.

Denial of Service is the process of denying service to an authorised user. For example, deliberately taking the server offline that is hosting the Bluetrak database that all clients are reporting to, this would make Bluetrak stop functioning. This is also referred to as availability.

Elevation of Privilege is when a user gains more privileges than entitled. In relation to Bluetrak this could be the user gaining 'Administrator' (or root) access if Bluetrak Client crashed (deliberately or accidentally) in a special way.

STRIDE categorises threats by their effect but we could also classify threats by their causes. It is also important to note that these threats are interdependent - unauthorised information disclosure could lead to tampering with data, etc.

Another method of organising threat analysis is to create threat trees (also known as attack trees). We look at our assets and start with the threats that we are most concerned about. While doing this process we need to think like an attacker and try to reach our target, which may involve sub goals. For example, we must first spoof an IP address in order to gain unauthorised access to the database. From this we are then able to create a list of countermeasures to put in place to stop these sub goals which in turn stop the initial goals of the attacker. This is in an essence risk analysis.

Risks are a measure of the damage an attack may cause and the likelihood for it to happen. Numerous methodologies exist to calculate risks; from the value of assets, the ease to exploit a vulnerability and the likelihood of the threat existing - which are the area of risk management and risk assessment in general.

3.3.2 DREAD

Risk assessment for software is done in much the same way but Microsoft's **DREAD** methodology is used. While using DREAD, risks are rated according to *Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability*.

Damage Potential refers to how great is the damage and what is the value of the data affected?

Reproducibility refers to how easy it is to get the attack to work. For example, does the attack use a feature installed by default or will it only work in some environments and not others?

Exploitability refers to how much effort and expertise is required to mount an attack. Is the cost too great for this to ever happen? Can the attack be scripted?

Affected Users refers to how many users would be affected if the specific attack occurred. Does the attack affect clients or just the server? Again, does the attack only work in special configurations and environments?

Discoverability refers to will the vulnerability be discovered? It is an extremely safe and logical option to assume yes it will be. If we discovered it, the chances of someone else finding it too are extremely high. Is it worth the risk?

For each category we would assign a value between 1 and 10 and the average over all the five values gives us the risk rating.

For example (and on a brief level in relation to Bluetrak), spoofing a Bluetooth tag's MAC address. The "Damage Potential" could be huge as things can be damaged anonymously as they are spoofing an authorised person. No one would find it unusual for "John Smith" of "Accounts" to be in the Accounts department - of course, this would be spoofed. Therefore, for "Damage Potential" that would be a 10. For "Reproducibility", anyone with enough technical knowledge can reproduce this attack. It can be done on a normal computer by editing the registry values of the Bluetooth USB dongle - it doesn't even need to attack the actual Bluetooth tag, so 8. "Exploitability" is much like "Reproducibility" in an essence - technical knowledge is required but not much. The user has to sniff MAC addresses in use and just reproduce these, again, 8. "Affected Users" is potentially everyone although the real person getting spoofed is probably the one that will be the first port of call if any damage occurred, so a 7. "Discoverability" remains extremely high at 10 as it's logical to think this attack will work especially if an attacker is aware of how Bluetrak works.

$$10 + 8 + 8 + 7 + 10 = 43 / 5 = 8.6$$

8.6 is the overall risk level for spoofing MAC addresses of Bluetooth tags. This is extremely high. Processes and controls will obviously have to be put in place to reduce or mitigate this risk for Bluetrak.

Recently there has been much discussion that DREAD is an outdated way of threat modelling¹, namely by Howard and Lipner [MH06] of Microsoft itself. This is mostly due to the scores being assigned are too subjective since we have business and technical type security consultants giving different values based on their knowledge and expertise. Microsoft are now using something different and that is their “Security Bulletin Severity Rating System” from “Microsoft Security Response Center.”² Instead of having a rating system between 0 and 10 where the majority of threats are ranked at either a 1 or a 10, it is now broken down into 1 or 4 categories.

1. **Critical:** A vulnerability whose exploitation could allow propagation of an Internet worm without user action.
2. **Important:** A vulnerability whose exploitation could result in compromise of the confidentiality, integrity, or availability of user’s data, or of the integrity or availability of processing resources.
3. **Moderate:** Exploitability is mitigated to a significant degree by factors such as default configuration, auditing, or difficulty of exploitation.
4. **Low:** A vulnerability whose exploitation is extremely difficult, or whose impact is minimal.

¹ “DREAD is dead” http://blogs.msdn.com/michael_howard/

² Security Bulletin Severity Rating System:
<http://www.microsoft.com/technet/security/bulletin/rating.msp>

Many organisations and businesses have adopted this new style of rating risks during threat modelling as it gets to the heart of what the threats (and their relevant risks) are and how they should be viewed in relation to the organisation as a whole.

3.3.3 Responding to Threats

After risk analysis has taken place and we are aware of what threats target our application, the next step is to respond to these threats, obviously. However, as a developer these choices vary depending on the actual stage that we have considered security in our application. The later security is left (or none at all) in the development of an application, the greater the consequences are and vice versa with the opposite results.

One could do nothing but its only going to be a matter of time before someone else finds the vulnerability and submits it to a security list like Bugtraq. This would be rather embarrassing, especially if it was known that the vendor was aware of the vulnerability but did nothing. Another option is to warn the user. For example, let the user know that by doing a specific action that they are taking a risk; "Do you really want to turn off your firewall? This will possibly allow attackers to compromise your machine." However, this will not stop that threat as users are rarely qualified to make such decisions - it is merely a disclaimer for the developer so they can state, "We did warn you!" If a vulnerability was found during the design stage then the logical option would be to remove the security bug from the product altogether. Again, this is not as simple as it sounds but is easier to carry out as the product has not yet been released to the public. Trying to remove a feature that people are currently using is going to be an extremely difficult task. The final option and probably the most logical option from a human perspective is to fix the problem. This comes in the form of security patches, updates, etc. Although, again we are relying on the end user to install these patches - something Microsoft has got the hang of with their "AutoUpdate" but may not be so easy for other developers.

Security is a design and an implementation issue and it is rarely possible to address all security issues at the design stage. However, if we do not attempt to consider security at all in the design stage then there are going to be problems later on down the line. Refinement often doesn't work in security, that is why it is so important that developers think with a "security brain" when creating new applications. This can be achieved by developers using the Security Development Lifecycle and replacing all outdated Software Development Lifecycles which do not consider security all.

If Bluetrak had implemented a Security Development Lifecycle and followed it then "Testing" and "Code Review" stages would be the next step to progress onto. However, the only testing that has been carried out in Bluetrak is functionality testing, i.e. does this work, if so, why not? This occurred during the development stage while the application was being coded. Of course this means that the security assessment isn't going to be anywhere as effective (as discussed earlier) as it has no "Abuse Cases" to work from.

Chapter 4

Security Assessment

In this chapter, we carry out a Security Assessment of Bluetrak. We discuss what the implications of not following a Security Development Lifecycle during the development are to the Security Assessment. We discuss what would (and should) happen usually if one had been followed, and what we are to do instead and the consequences of this alternative route. We apply what we have learnt from previous chapters in terms of Risk and Threat Analysis. However, we discuss why they will not be as effective because they are being applied at such a late stage. In this chapter it will become obvious what happens if you do not consider security during the development of an application (or consider it too late), from the perspectives of both the developer and the security consultant carrying out the Security Assessment.

4.1 Security as an Add-on?

A Security Assessment is a form of testing; however it is solely looking at the application from a security perspective. Assessing the security of the application - assessing the processes and controls put in place to mitigate the security threats and risks posed to an application. If we had developed Bluetrak from the start with security in mind (such as using a Security Development Lifecycle (SDL)), then this wouldn't be the first time that its security had been tested. However, since Bluetrak hadn't followed a SDL, this is the first time security would have been considered. This could mean after the application has been developed, or even worse, after it has been shipped and is in use by the public. The consequences of only considering security at both of these different stages were briefly discussed in the previous chapter.

The first thing we would do is to refer to the initial “Abuse Cases” documented during the initial design stage when threats to the application were discovered. This was documented even further during the Threat Modelling stage. Within these “Abuse Cases” contained how to carry out these attacks and preventative measures (within the Security Requirements stage) that would need to be implemented, and probably have. As such, we would only be testing how effective the processes and controls that had been put in place were. However, we have a problem. No such “Abuse Cases” exist - which means no security controls or processes are in place. We are left with the job of performing the task of identifying threats, testing for them (including analysing code), analysing the risks of these threats and suggesting preventative measures. Additionally, another security assessment will need to be performed at a later date to assess the effectiveness of processes and controls that need to be put in place.

Where do we start? That’s a good question and a huge one at that because we have no indicators (like the “Abuse Cases”) to start from. It’s far too late to implement a Security Development Lifecycle (SDL) now. What we can do though is partially implement bits from it. However, this won’t be nearly as effective as if we had considered a SDL right from the start. This Security Assessment will take a lot longer than normal as we are carrying out what should have been done during the design and development of the application. This will of course take time, and as every developer knows, time is money.

What is happening here is that security is being treated as an add-on. This is not the fault of the security consultant carrying out the security assessment - he/she has no other choice, other than to ask the developer to redo the application using a SDL; which isn’t going to happen (not this time anyway).

We are carrying out a “penetrate-and-patch” approach to software development which McGraw [GM99] states exactly how not to design secure

applications. Because security has not been considered from the start, we have no other choice. However, we can improve this approach by carrying out a little of what should have been done before - "Threat Modelling", "Risk Analysis", "Code Review", etc.

4.2 Bluetrak Threats and Risks

A good place to start would be to create some Abuse Cases to discover what threats might exist in Bluetrak. To recap from Chapter 3, Abuse Cases focus on the developer considering deliberate misuse of the application, different scenario's; buffer overflows, malicious data, etc. and how the application responds to these. However, our Abuse Cases will differ slightly as we are short on time (having to conduct this abnormally sized security assessment post development) therefore we will have to test for these scenarios (based on threats) at the same time. Additionally, we will be carrying out a Risk Analysis - possibly missing out things that would have been spotted if we had followed a SDL as the Abuse Cases are looked at again in different stages and touched on more with greater detail.

As documented in the previous chapter, a good way of discovering (or categorising) threats (or possible attacks) to an application is by using the **STRIDE** methodology. Microsoft's **DREAD** methodology will be used within these threats to rank them in terms of Risk.

4.2.1 Spoofing Identity

Bluetrak relies on MAC addresses for identification (and accountability), that's how the whole concept of Bluetrak works. If an attacker managed to spoof a legitimate (valid) MAC address of a Bluetooth tag belonging to another person then Bluetrak would assume (wrongly) that they are the identity of this person. This would mean that people are wrongly identified and this would obviously provide no accountability at all.

It is important to note that this attack is not aimed at the Bluetooth tag directly. An attacker can use a computer attached with a USB Bluetooth

dongle. This attack would be carried out by the attacker being in the environment that Bluetrak is running and collect (or harvest) all discovered devices noting down their MAC addresses. By carrying out this attack in the environment that Bluetrak is running, the attacker can be sure that these Bluetooth devices (with their relevant MAC addresses) are in use by Bluetrak and are simply not rogue devices; i.e. mobile phones, etc. There is of course a window for errors to occur by the attacker. There is the possibility that the attacker could be spoofing the MAC address of a mobile phone, which would show up as not being owned by an authorised person in the Bluetrak Server application. However, if common sense is applied by the attacker and he/she is vigilant around their surroundings while new devices are being found then he/she can guess with a high probability who owns what MAC address. For example, the attacker is sitting in Reception, no devices are found. Suddenly a device shows up as a man in a suit walks into Reception and then swipes his ID card and continues into the Accounts department. It would be safe for the attacker to assume this man is an employee and that he probably belongs to the Accounts department. Of course, this man could be a visitor. However, if the attacker used his common sense (was the man wearing a visitor's badge?) he/she could soon gather a collection of valid and useful MAC addresses to spoof later. Useful referring to not only spoofing the identity of that person but spoofing entities (or credentials) related to them too. Again, for example, if the attacker wanted to gain access to the Accounts department, by using the MAC address of the man that just walked in, it wouldn't be unusual for that person to be in Accounts as he belongs to that department. However, if this person was suddenly in Sales, whoever watching the Bluetrak Server application may be slightly concerned as he doesn't belong to this department. This is a prime example of intelligent, useful spoofing by using common sense.

This threat poses a huge problem for Bluetrak. Therefore we need to use Microsoft's **DREAD** methodology within this threat to work out what risk it poses.

The “Damage Potential” could be huge as things can be damaged anonymously as they are spoofing an authorised person. No one would find it unusual for “John Smith” of “Accounts” to be in the Accounts department - of course, this would be spoofed. Therefore, for “Damage Potential” that would be a 10. For “Reproducibility”, anyone with enough technical knowledge can reproduce this attack. It can be done on a normal computer by editing the registry values of the Bluetooth USB dongle - it doesn’t even need to attack the actual Bluetooth tag, so 8. “Exploitability” is much like “Reproducibility” in an essence - technical knowledge is required but not much. The user has to sniff MAC addresses in use and just reproduce these, again, 8. “Affected Users” is potentially everyone although the real person getting spoofed is probably the one that will be the first port of call if any damage occurred, so a 7. “Discoverability” remains extremely high at 10 as it’s logical to think this attack will work especially if we are aware of how Bluetrak works.

$$10 + 8 + 8 + 7 + 10 = 43 / 5 = 8.6$$

8.6 is the overall risk level for spoofing MAC addresses of Bluetooth tags.

4.2.2 Tampering with Data

Bluetrak works by having a central database to which client nodes generate reports and send them to. This central database concept is great in terms of the functionality for Bluetrak; however, whenever one central place exists in terms of information security, that central place becomes the focus of an attacker. This is where the information is held that an attacker wants to get hold of. The subject of database security is an entire project in itself (of which is out of the scope of this project). The database uses Microsoft Access 2002. Access has features such as Access Control Lists (ACL) - a Discretionary ACL (DACL), password protection and most importantly workgroups; user and group accounts. Using features like these allow one to set permissions and ownership on various objects within a database. Information about workgroups and accounts are stored in a workgroup

information file; SYSTEM.MDW. This file contains a SystemID (SID) which is essentially a binary code which identifies each user and contains what permissions they have. For example, "USER A" can only read from the database and not write, whereas "USER B" can read and write, or "USER C" can only write but not read. This again is the subject of information flows and access control matrices. The SYSTEM.MDW file contains three pre-defined accounts:

- Admin - The default user account. This is the same for every installation of Microsoft Access (installed by default).
- Admins - The administrator's group account. This can contain a collection of user accounts so when a permission is assigned to a group, it is also applied to all the users contained within.
- Users - The group account containing all user accounts.

When Microsoft Access is first installed, the user is automatically made a member of the group named "Admins" with a user account called "Admin". By default, this contains an empty password and a Personal Identifier or Personal ID (PID). The security problems of doing this are of course obvious. A user is not made aware of any multi-user feature of Access until they activate the feature or if they go and look for it intentionally. Therefore, the average user will not be aware of the security implications of this.

Bluetrak does not take advantage of these features and uses the default "Admin" account. This is done for a variety of reasons, mostly due to functionality and cross platform support. A client node does not require Microsoft Access to actually be installed on their computer to be able to interact with an Access database file. Microsoft Windows allows this to happen via the MSJET engine. This means that client nodes within the Bluetrak network can be running any version of Microsoft Windows and still interact with the database. However, the only way that this can be done effectively is by disregarding the feature of multi-user support, which in turn we are removing the ability to assign permissions. Some ODBC (open

database connectivity) drivers in older versions of Microsoft Windows have compatibility issues with this and Microsoft Access 2002. Therefore the default "Admin" account is used by not defining any specific username when opening the database.

Access to the database requires no credentials to authenticate at all. However, because of the way Bluetrak works in connecting to the database, these credentials would simply be in plaintext in the `btClient.exe.config` and `btServer.exe.config` XML files anyway. At first glance, no authentication sounds like complete madness. However, Bluetrak implements security a little bit further down the stack. Bluetrak uses Microsoft Windows' Networking "File and Printer Sharing" (NETBIOS) feature as a form of security because all client nodes are remotely connecting to the database on another computer. The central database on the remote computer is shared and is also password protected. Additionally, permissions (a Discretionary Access Control List - DACL) are set here based on different users (read, read/write, execute). This form of security can be applied locally at the host computer or for much more advanced control, via the domain controller within the network via Microsoft's Active Directory. The later option is more secure as NETBIOS (Windows Sharing) has had its fair share of security problems. This was mainly down to it being trivial to brute force weak passwords assigned to shares by users but later a vulnerability (CVE-2000-0979) was discovered that allowed shares to be accessed without supplying a password at all¹. This initially only affected Windows 95, 98, 98 SE and ME and a patch is available from Microsoft that fixes this. Microsoft has improved on this issue since in Windows 2000 and XP (and Vista to come) with the use of Kerberos² for authentication. Bluetrak is designed for use with Windows XP (SP2), although older versions of Windows (with Microsoft's Bluetooth stack installed) will run it.

¹ 'Share Level Password' Vulnerability (CVE-2000-0979)
<http://www.microsoft.com/technet/security/bulletin/MS00-072.msp>

² "Kerberos: The Network Authentication Protocol" - <http://web.mit.edu/Kerberos/>

Bluetrak is therefore relying on Microsoft Windows for its security, which could be argued is a good or bad thing.

Even with taking into account the above, the "Damage Potential" for this threat would undoubtedly be a 10 (risk rating). If the data is tampered with (in an unauthorised manner) then identification and accountability; the goals that Bluetrak was to provide are no longer provided. If the host computer sharing the database is not running at least Windows XP (SP2) or 98/ME with patches then it would be trivial for an attacker to launch a dictionary brute force attack on the shares. However, if the host computer is running Windows XP then it is patched against this vulnerability by default. Therefore, "Reproducibility" is at a 6. This value remains high because we are still relying on the operating system (OS) to implement and maintain the security controls and processes it has in place. Vulnerabilities occur in operating systems everyday of the week, some are of high severity, some of little or no importance at all. However, it would be wise for anyone running Bluetrak to monitor security mailing lists such as Bugtraq to keep up to date and patch immediately if a security hole is found in the OS. It is important to note that this will not guard against "0-day" (zero day) attacks to which the vendor is not aware of. This is a problem for computer security as a whole, not just related to Bluetrak. We cannot guard against what we do not know about - which seems only logical. This relates to "Exploitability". We do not know of how easy/hard it could be to exploit a vulnerability in the OS. The scope is so wide that it would be impossible to say. The only thing we can do is base our risk on the present moment in time, which of course will have to keep being reviewed. Therefore, for "Exploitability" the risk is rated at 6, again, this could change at any given minute depending on the risk ranking "Microsoft's Security Response Center" assigns on their operating system. If an attack at bypassing the security controls of the OS was successful then of course this would affect all users of Bluetrak. The attacker could cause absolute mayhem with Bluetrak if he/she tampered with the data or deleted the database entirely (covered in Denial of Service). "Affected Users" is ranked at a risk rating of

10. "Discoverability" will also remain high at 10 because it is good security practice to not underestimate an attacker and assume things about him/her. In relation to such a vulnerability being discovered, the chances are extremely high and this exploit could end up on a mailing list or remain "0-day". Either way, it is safe to assume the very worst case scenario; that a vulnerability like this could be discovered as the scope is so large - a security flaw in Windows which would allow an attacker to circumvent (or compromise) authentication and authorisation.

$$10 + 6 + 6 + 10 + 10 = 42 / 5 = 8.4$$

8.4 is the overall risk level for tampering with data in the database that Bluetrak relies on via compromising the operating system.

4.2.3 Repudiation

One of Bluetrak's goals is to provide accountability (which is repudiation). If a person (an employee) denies being in a specific area, Bluetrak will attempt to prove otherwise with its audit log of reports generated by the client nodes. These reports contain more detailed information such as the time, date and most importantly the location of all the person's activities for that specific time period. The only two factors that could dispute this would be that the person's MAC address of their Bluetooth tag had been spoofed, or, that the database has been tampered with. The database in itself has its own audit log provided by Windows so this also adds to Bluetrak's accountability claims on another level. Therefore, the only real concern in not being able to provide accountability would be if an attacker had spoofed their MAC address. However, we cannot rule out tampering with the data completely. The risk will be calculated by looking at the "Spoofing Identity" and "Tampering with Data" threats as the two are linked to this threat and coming to a compromise between the two.

"Damage Potential" for both is a 10, therefore will remain this. "Reproducibility" is an 8 and 6 respectively, therefore will be assigned a 7.

“Exploitability” is again an 8 and 6, so will it will become a 7. “Affected Users” is a 7 and a 10, therefore will become 8.5. “Discoverability” for both is a 10, therefore will stay at 10.

$$10 + 7 + 7 + 8.5 + 10 = 42.5 / 5 = 8.5$$

8.5 is the overall risk level for Bluetrak not being able to provide repudiation (accountability). The use of cryptographic keys could fix this.

4.2.4 Information Disclosure

Information Disclosure is also referred to as confidentiality and in relation to Bluetrak is linked to the “Tampering with Data” threat. Although Bluetrak’s goal is not to provide confidentiality, if unauthorised access to the database was granted then the consequences would be massive. Because of the way Bluetrak does not implement authentication or authorisation at the database level (no permissions within the database) - it is more at risk of disclosing information. As stated earlier (in the “Tampering with Data” threat), Bluetrak is relying on the operating system (OS) to implement security controls in place via Microsoft Windows’ Networking File and Printer Sharing. If an attacker managed to gain access to the database then he/she would not just be able to view information but would also be able to manipulate and change it, in an unauthorised manner. The only feature that would stop this would be to implement permissions lower down the stack and assign different users groups either in the share properties itself or via Active Directory. Again, this would still be relying on the OS to take care of the security and as stated earlier, there is no ruling out the OS getting compromised by an attacker.

This is closely linked with “Tampering with Data” as it shares the same properties in terms of threat, therefore it will mirror the risk rating for it.

“Damage Potential” is a 10, “Reproducibility” is a 6, “Exploitability” is a 6, “Affected Users” is a 10 and “Discoverability” is also a 10.

$$10 + 6 + 6 + 10 + 10 = 42 / 5 = 8.4$$

8.4 is the overall risk level for Bluetrak disclosing information to unauthorised parties via either a brute force dictionary attack at the shares on an un-patched computer hosting the database, or by an attacker compromising the OS of the computer the database resides on.

4.2.5 Denial of Service

Denial of Service is when an authorised user is denied service to a resource. In the case of Bluetrak, the “user” is the client node (be it the client or server application) and the “resource” is the central database. If the client nodes are unable to write to the database then Bluetrak will not work. If the attacker’s goal was to stop Bluetrak functioning then all he/she would need to do would be to stop communication between the nodes and the central database. This is something that was stated in the “Tampering with Data” threat; the attacker will focus on this central point which turns out to be a central point of weakness.

An attacker could carry out this attack in many ways; by crashing the computer hosting the database, a variation of flooding (DoS) attacks (Ping, Smurf, Teardrop, Fraggle, etc.) These could be of a DDoS (distributed denial of service) nature. It is important to note that the environment that Bluetrak will be running in will be that of an internal network (an intranet), not directly connected to the Internet. Therefore the risks of such attacks are significantly lowered, however, one should not rule out the possibility of an attack originating from inside the network. By default, Windows XP firewall is enabled and will stop such attacks as Ping floods, however, will not guard against spoofed TCP/UDP DDoS attacks (such as Smurf) and SYN/ACK flooding. The computer would also be vulnerable to man-in-the-middle attacks such as ARP spoofing. ARP spoofing would fool the client nodes into thinking the attacker’s computer is the computer that hosts the central database. If the attacker created a database with the same name

and mirrored the share names, he/she would then be able to steal reports from the client nodes and no one would be the wiser.

The ARP spoofing attack threat should really belong in the "Spoofing" section. However, it has only been discovered during this "Denial of Service" threat section. This demonstrates how easy it is to overlook things when trying to perform threat modelling after development of the product and only at the security assessment stage. The whole concept of the attacker being able to remodel the TCP/IP stack has been overlooked. If a Security Development Lifecycle (SDL) had been followed then there would have been a very low possibility of this being overlooked as the "Abuse Cases" are looked at again and again and added to. However, the development of Bluetrak didn't and this is an example of how hard conducting a security assessment in this way has become. This will of course have to be reviewed again later in more detail.

If a Denial of Service attack is allowed to occur, Bluetrak will no longer function. In terms of risk, "Damage Potential" is ranked at a 10 for this obvious fact, be it long or short time damage to availability. "Reproducibility" is also a 10 - it is trivial to carry out many of the previous attacks described, especially with the amount of tools readily available (it can be scripted, and thus, "script kiddies" are huge in numbers, huge risk). "Exploitability" is assigned a 10 because "script kiddies" with no prior knowledge of what they are doing (or how they are doing it) can carry out these attacks. "Affected Users" is everyone, therefore a 10. "Discoverability" is a 10 also, it is well documented where to get these tools from, what attack vectors to aim at (discussed later) and how to carry out these attacks.

$$10 + 10 + 10 + 10 + 10 = 50 / 5 = 10$$

10 is the overall risk level for a Denial of Service attack to occur that would stop Bluetrak functionally. This risk level may seem a little extreme if we

think back to the fact that we have Windows XP Firewall enabled by default which will (with the right rulesets) stop the majority of these attacks. However, these rulesets are not going to be put in place by default and hence we must expect the worst case scenario. With a firewall in place on the host computer then this risk is reduced significantly. However, it should be noted that a DoS attack need not involve a computer or an attacker at all. It could be of a physical nature; accidental or deliberate damage to the communications link between the nodes; network cable, switch, hub, etc.

4.2.6 Elevation of Privilege

An Elevation of Privilege threat does not really apply to Bluetrak. Bluetrak is simply a reporting tool that is security related. It does not require any special privileges (it is not a system program), nor does it function as any security control or process such as an access control. It is related to security, however not directly. This is extremely nice to hear since all the other threats have extremely high risk levels associated with them. An Elevation of Privilege could still occur in the operating system, but this would not be any more aided than normal than with Bluetrak. Therefore buffer overflows and attackers inserting shellcode are really the least of our worries. Obviously they are not ok as the program would segfault (crash) within itself and could leak memory. However, we can rest assured that if Bluetrak crashed abnormally, it would not drop into an administrator (or root, uid=0/gid=0) account with special privileges. Additionally, the computer hosting the database should be locked (out of the trusted path - secure attention sequence logon window) at all times. Therefore, if a client node computer is compromised, this affects the central database in no way.

There, we really only have the operating system to go on in terms of being vulnerable to an Elevation of Privilege attack. This attack would really have to be "local" to be effective at all. "Damage Potential" is a 5, "Reproducibility" is a 3, "Exploitability" is a 3, "Affected Users" is a 5 and "Discoverability" is also a 3.

$$5 + 3 + 3 + 5 + 3 = 19 / 5 = 3.8$$

The overall risk level for an Elevation of Privilege threat becoming real is unlikely, however, cannot be ruled out. We are relying on the operating system and thus this risk is really out of our hands.

4.3 Summary of Bluetrak Threats and Risks

Table 4.1: Threats (from STRIDE) and their Risk Level (from DREAD)

Threats	Overall Risk Level	Microsoft's Security Bulletin Severity Rating System
Spoofing Identity	8.6	Important
Tampering with Data	8.4	Important
Repudiation	8.5	Important
Information Disclosure	8.4	Moderate
Denial of Service	10	Important
Elevation of Privilege	3.8	Low
TOTAL OVERALL RISK:	47.7 / 60	

47.7 out of a possible 60 risk 'points' is an extremely high figure for Bluetrak. This is due to the fact that the first time threat modelling has taken place is after the application has been developed. These risks would still exist if threat modelling was in fact carried out at the design stage (by using a SDL). However, processes and controls would be in place to respond to these threats and to mitigate or reduce the risks they pose. If that was the case then we would be looking at a completely different set of figures, figures which wouldn't be as worrying. By carrying out a risk and threat analysis at this late stage, all we can do is give feedback to the developer or advice on what he/she should do to fix this. We (as security consultants carrying out the audit) cannot put these controls in place and review them as it would be a conflict of interest - we would be reviewing

our own work! Therefore the results from the security assessment will need to contain and provide vital information for the developers and their team to enable them to fix these problems.

4.4 Code Review

Code review is important in any application's development. However, a code review with security in mind is slightly different. The developer looks at the code with functionality in mind whilst the security consultant looks at that same code with security in mind. If the developer has not been educated about secure coding then he/she obviously wouldn't know what they were looking for in respect of security bugs. At the same time, the security consultant is a security consultant first, a programmer (we hope) second, and therefore he/she might not have a clue about the application in terms of functionality. This goes back to the gap between Software Development and Information Security discussed in Chapter 3. However, both groups of people need to talk to each other in relation to their different approaches if things are to change. This code review would have occurred early on in the development stage right after the initial design stage, if a Security Development Lifecycle (SDL) had been followed. Of which as we know, Bluetrak did not follow. That is not to say the developer has not carried out testing - functionality testing will have been carried out else the application wouldn't be working. We are now left with the task of reviewing the code from a security perspective. However, we will have to feed back to the developer our findings in a way that educates him/her at the same time. This security assessment is already occurring after the application has been developed and not during. Therefore it will require motivation by the developer to go back and fix these things.

As Bluetrak is not going to be a system program of any kind and thus will not require any system privileges, we are not too concerned about buffer overflows leading to elevation of privileges. This threat has been covered already and came out with an overall low risk level. However, if the program did crash abnormally then memory leakage could occur which

would lead to information disclosure. Therefore we still need to check for these types of bugs in the code.

The two most common programming mistakes concerning security are buffer overflows and format string vulnerabilities according to Howard, LeBlanc, et al. [MH05].

Buffer Overflows

A buffer overflow occurs when a program (or application) attempts to store more data in a buffer than it was originally designed to hold. The data of course has to go somewhere and overflows into memory, be it adjacent buffers or other addresses in memory. This will overwrite whatever was being stored at these locations (for example pointers) to which the program is going to refer to concerning what instruction to execute next. This is almost always a mistake by the programmer and will lead to attacks on data integrity and information disclosure (in terms of memory leaks). This can of course allow an elevation of privilege attack to occur when carried out properly and deliberately. The attacker is able to overflow the buffer in such a way that he/she can calculate where in memory they are overwriting (usually the return address) and insert their own instructions (machine code, usually shellcode). The program will then treat the instruction at this memory location just like any other instruction initially coded by the programmer. However, it wasn't created by the programmer, the attacker created it, but the program cannot tell the difference. The program will then go ahead and execute this instruction. The attacker is controlling (and changing) the flow of execution. This is extremely dangerous because if the program runs with system privileges, then the attacker now has control of a system program and can make it do what he/she wants it to do - usually execute a root shell. Computers cannot tell between data and instructions (code) since they are both contained on the same stack, this is where the problem lies; of which Trusted Computing and various other methods try to fix. Methods other the years have included non executable stacks and the use of canaries. Non executable stacks provide a separate stack for data

and code (or instructions). Therefore if a buffer is overflowed (which is held on the data stack), then even if machine code (shellcode) is inserted, it will not be executed or treated as code as the computer knows this is data. Canaries are simply a protection added to different memory locations with a checksum to check their value is what it is supposed to be. If a return address is changed for example, the canary will spot this and cause an error (exception error). This name comes from when miners used to use canary birds as warning signs if gas was about.

Format String Vulnerabilities

Although format string vulnerabilities are not as common as buffer overflows, they are still as dangerous in terms of security. Format strings are used to control the formatting of I/O (input/output) within C/C++ applications. They contain special identifiers (%s for strings, %d for integers, etc.) that if used maliciously when concerning input, can reveal dangerous information about what is on the stack and variables used by various functions. The %n identifier can be used to overwrite data in memory. This gives the same outcome as buffer overflows; allowing the attacker to control execution flow of the program and get it to execute what they want (arbitrary code execution). The cause of format string vulnerabilities is again a programming error and is to do with the use of variable argument functions in C/C++. For example, the programmer using `printf(string);` when `printf("%s", string);` should be used.

4.4.1 Code Analysis

Because we have left this code review so late, the question of where to start haunts us again. Bluetrak is an extremely large application; hundreds of lines of code exist. Different functions exist such as the Client Application, the Server Application, the API allowing the Client Application to speak to the Bluetooth USB dongle and finally the API allowing both to access and manipulate the database. We do not have time to simply go through the code line by code manually. Therefore an automated tool is ideal for drawing our eyes to security problems in the code, far quicker than

we ever could. RATS¹ (rough auditing tool for security) is ideal for this. As its name implies, RATS performs only a rough analysis of source code. It will not find all errors and may also flag false positives. However, RATS will serve the purpose of saving us time but by not cutting corners - its vast database of vulnerabilities will ensure nothing silly or trivial is missed.

RATS was ran on all the files to see if any vulnerabilities contained within its vulnerability database matched those in Bluetrak's code. It should be noted that not everything can be found and that the auditing tool is only as good as the database of bugs on which it relies upon. We do have an advantage in the fact that the majority of Bluetrak is coded in C# .NET. C# is a type safe language, which means it guarantees the absence of un-trapped errors. Safety is guaranteed by static checks and by runtime checks. C# also carries out automatic array bound checking and garbage collection. Therefore our only real concern is C and C++ files, of which the btInfo.dll is written in.

The RATS test resulted in the following:

Entries in perl database: 33

Entries in python database: 62

Entries in c database: 334

Entries in php database: 55

Analyzing d:\bt06\btApp\btInfo\btInfo.cpp

d:\bt06\btApp\btInfo\btInfo.cpp:70: High: fprintf

Check to be sure that the non-constant format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle.

d:\bt06\btApp\btInfo\btInfo.cpp:188: High: sprintf

Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain '%s' without precision that could result in a buffer overflow.

¹ RATS: <http://www.securesw.com/rats/>

Total lines analyzed: 296

Total time 0.000000 seconds

0 lines per second

As stated, RATS was run on all the files (including even the C# ones) just to be sure. As predicted, the btInfo.dll file (btInfo.cpp pre-compiled) is the only one that RATS is concerned about. It has noticed that a format string vulnerability could exist here and has advised us to check the origin of the data going into this.

If we look at line 70 in btInfo.cpp concerning the “fprintf” function, it is as follows:

```
SYSTEMTIME stLocal;
GetLocalTime(&stLocal);
fprintf(stream, "%02d/%02d/%d %02d:%02d:%02d.%04d\t",
stLocal.wDay, stLocal.wMonth, stLocal.wYear,
stLocal.wHour, stLocal.wMinute, stLocal.wSecond, stLocal.wMilliseconds);

fprintf(stream, buf);
fclose(stream);
```

The data that is going into the “fprintf” function has been taken from SYSTEMTIME (stLocal) and not from user input. Therefore there is no possibility that an attacker could take advantage of this. If this had been taken from user input then this would be an area of worry and would have to be recoded so that the input was parsed to some kind of validation system to remove any attempts by an attacker to manipulate the format string.

```
sprintf(debugBuf, "**API CALL* | after WSALookupServiceBegin(): iResult = %i,
hLookup==NULL ? %i\n", iResult, hLookup == NULL);
debugInfo(debugBuf);
```

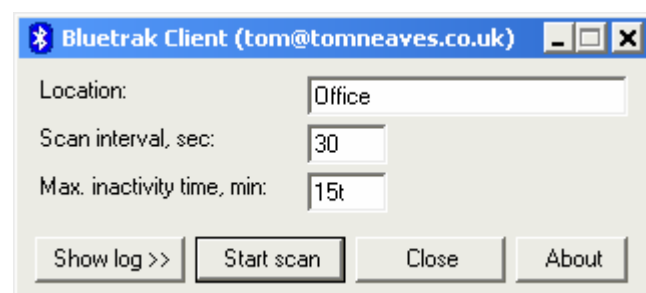
Line 188 concerning the “sprintf” function is much the same. It does not generate its data from user input and additionally, this is only used for

debugging purposes (it is only active when `DEBUG=1`). It could be taken out altogether. However, if a problem ever occurred in the code without a debugging feature existing, things would get rather difficult for the programmer who is trying to track down what is going wrong.

4.4.2 Never Trust your Inputs!

As we've seen, any programming errors that occur have security consequences because the user is able to take advantage of them in the form of user input. A buffer overflow occurs when a program takes in an input from the user and attempts to store it into a fixed size buffer. This buffer is too small and does not fit everything in, thus the rest overflows. A format string vulnerability occurs when a program attempts to format values which have also been inputted by the user (string, integer, etc.), leading to deliberate manipulation. Therefore it is always good programming practice to never trust your inputs. We cannot predict what a user is going to input. However, what we can do is plan for the majority of possibilities the user can input, test for them and respond accordingly. This comes in the form of fuzz testing, data mutation, random (or unexpected inputs) and then implementing error handlers and validation methods to catch and stop these. Fuzz testing (or fuzzing) is the idea of attaching the inputs of a program to a source of random data. If the program fails (usually by crashing) then error handlers (case, try/catch, if/else if C statements) need to be implemented to fix this. This tests functionality along with security at the same time. An example of this is shown below on the Bluetrak Client.

Figure 4.2: Unexpected Input in Bluetrak Client



In relation to Figure 4.2 on the previous page: If we enter "15t" in the "Maximum inactivity time" field, something bad should happen. The program is only expecting data of numerical type to be entered here and we have just inserted an alphabetical value. If we click "Start Scan" we are then presented with an error message alerting us of this fact (see below).

Figure 4.3: Error Message due to Unexpected Input in Bluetrak Client



This is because we have implemented "try" and "catch" statements outside "if" and "else if" statements within the code concerning all these inputs.

```
try
{
if ( (tbLocation.Text = tbLocation.Text.Trim()) == "")
strErr = "Enter location!";
else if ( (tbScanPeriod.Text = tbScanPeriod.Text.Trim()) == "")
strErr = "Enter scan time!";
else if ( (scanPeriodSecs = Int32.Parse(tbScanPeriod.Text)) <= 0)
strErr = "Scan time is incorrect!";
else if ( (tblnactivityTime.Text = tblnactivityTime.Text.Trim()) == "")
strErr = "Inactivity time is incorrect!";
else if (Int32.Parse(tblnactivityTime.Text) <= 0)
strErr = "Inactivity time is incorrect!";
}
catch (FormatException ex)
{
strErr = ex.Message;
strErr = "Please enter correct number!";
}

if (strErr != "")
{
MessageBox.Show(this, strErr, "Wrong data", MessageBoxButtons.OK,
MessageBoxIcon.Error);
return;
}
```

Therefore, if an input does not match what we are expecting, we will not accept it and will display a message warning them of this and to input it

again correctly. This way, the data inputted by the user is not acted on (or processed) by the program which would cause errors. Instead it is acted on before the values even leave the textboxes and are copied into the relevant variables. This is data validation within Bluetrak, of which is very effective.

Programmers can use different methods for carrying out validation on user inputs. Some choose to use white lists, others use blacklists and some just parse user input and strip out certain characters before the input is acted on. There are numerous advantages and disadvantages of doing these various methods, although white lists seem to be the most problematic as the programmer is restricting what a user can type which may cause simple problems. For example, not allowing a "'" (single quotes) in a name field, when people with surnames like "O'Connor" may be using the application and it will not accept their name as a value. Blacklists have the disadvantage of an attacker being able to circumvent them by taking advantage of various features provided by the operating system, canonicalization support for example.

Even when security is considered during programming, things can still go horribly wrong. This is true when using a blacklist as a form of data validation.

Too much focus on security can get in the way of functionality. It is therefore a decision that the developers must make after being educated of the security consequences by their security people mainly in the form of a Security Assessment. However, there is a fine line (and balance) between functionality and security and some developers will want to take that extra risk in order to gain more functionality.

Chapter 5

Conclusion

It is apparent that a serious gap does in fact exist between Software Development and Information Security. The only way in which this can be fixed is with education. We have looked into the different ways of implementing Information Security into Software Development and discussed the advantages of doing so. We have also looked at how to do this effectively, and also how not to do it with the implementation of Bluetrak serving as an example. We have investigated into what the consequences of not considering security (or considering it too late are) when developing an application can be. These consequences became more than obvious during the late Security Assessment when we had to carry out bespoke solutions with what we had learnt before. It was also proved that by treating security as an add-on that lots of things were missed and overlooked; such as only realising an attacker could remodel the TCP/IP stack (replays, etc.), ARP spoofing, etc. later on in Threat Modelling. This would of course been discovered early on in the design stages (Abuse Cases) if a Security Development Lifecycle (SDL) had been followed.

The most disturbing factor resulting from treating security as an add-on were the results of the Risks Analysis. A total risk rating of 47.7 out of a possible 60 was achieved for Bluetrak. This is nowhere near acceptable. Although at first glance Bluetrak seems to provide identification and accountability, it cannot provide this reliably because of this high risk rating. Bluetrak should be redeveloped with security in mind, right from the start, or alternatively all these problems presented in the Security Assessment need to be fixed to reduce these risks. However, having stated that the overall risk level is high; these risks are only related to the operation of Bluetrak, which is very important to note. By running Bluetrak we are not increasing the organisation's overall risk or creating new ones at

all. This is down to the fact that Bluetrak is not being used as a direct form of access control. If it was, then of course anyone that implemented Bluetrak for this task would be creating serious risks for their organisation as a whole. The threats discovered and security issues highlighted in the Security Assessment only affect Bluetrak in terms of its aims and objectives of providing identification and accountability. The Security Assessment draws to the conclusion that the claims of Bluetrak providing these are in fact false because of the way it has been developed and implemented.

You cannot expect an application to be secure by treating security as an add-on and only considering it at much later stages in the development. The case study of Bluetrak hopefully emphasised this fact, along with the discussion on what should have been done throughout.

Bluetrak in terms of functionality and design works great. However, whoever runs Bluetrak needs to be aware of the underlying threats posed and risks associated with it. The user of Bluetrak needs to understand that Bluetrak is a useful tool to aid security, but shouldn't be used solely as a form of security, especially not as an access control. If a legal case went to court concerning a dispute with an employee, Bluetrak would not be able to provide 100% identification and accountability in this instance. As long as the user of Bluetrak realises this (and has taken it into consideration during their implementation) then it will serve as a very useful security program. If they do not, they may be in for a shock when they attempt to present logs from Bluetrak in a possible court case. Another important factor that Bluetrak seems to miss (or overlook) is how is someone able to tell the difference between a laptop in a box with a Bluetooth tag inside it and an empty box with a Bluetooth tag inside it? Also, many legal boundaries need to be overcome if Bluetrak is to be implemented in a real world environment; such as the Human Rights Act¹, Chapter 42 (1998) mostly concerning Article 8 (Right to Privacy) and the Data Protection Act² (1998).

¹ Human Rights Act (1998) - <http://www.opsi.gov.uk/ACTS/acts1998/19980042.htm>

² Data Protection Act, Chapter 29 (1998) - <http://www.opsi.gov.uk/ACTS/acts1998/19980029.htm>

References

[GM98] Gary McGraw, "Testing for Security during Development: Why we should scrap penetrate-and-patch", IEEE Aerospace and Electronic Systems, 13(4), pages 13-15, April 1998.

[GM99] Gary McGraw, "Software Assurance for Security", IEEE Computer, 32(4), April 1999.

[GM05] Gary McGraw and Kenneth R. van Wyk, "Bridging the Gap between Software Development and Information Security", IEEE Security & Privacy, pages 64-68, September/October 2005.

[GM06] Gary McGraw, "Software Security: Building Security In", Addison-Wesley Professional, January 2006.

[MH02] Michael Howard and David LeBlanc, "Writing Secure Code" (Second Edition), Microsoft Press, Chapter 4, April 2002.

[MH05] Michael Howard, David LeBlanc, John Viega, "19 Deadly Sins of Software Security" (First Edition), McGraw-Hill Osborne, July 2005.

[MH06] Michael Howard and Steve Lipner, "The Security Development Lifecycle", Microsoft Press, June 2006.

Additional References

Christopher Alberts and Audrey Dorofee, "Managing Information Security Risks", Addison-Wesley, July 2003.

Jack Koziol, David Litchfield, et al., "The Shellcoder's Handbook: Discovering and Exploiting Security Holes", John Wiley & Sons, April 2004.

Appendix

<u>btInfo.cpp (btInfo.dll)</u>	<u>74</u>
<u>btDAO.cs (btDAL.dll)</u>	<u>79</u>

btInfo.cpp

```
#include <stdio.h>
#include <objbase.h>
#include <TCHAR.H>

#pragma comment(lib, "ws2_32.lib")

// set to 1 to enable Log file generation
#define DEBUG 0

void debugInfo(char*buf);

// how many time inquire devices per GetBluetoothDevices call.
// each time takes around 7 seconds!
const ULONG CXN_MAX_INQUIRY_RETRY = 1;

// main function
BOOL APIENTRY DIIMain( HANDLE hModule,
                      DWORD ul_reason_for_call,
                      LPVOID lpReserved
                      )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
            debugInfo("\n\n *****\n*START* | btDII is loaded...\n\n");
            break;
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
            debugInfo("\n\n*STOP* | btDII is unloaded...\n\n *****\n\n");
            break;
    }
    return TRUE;
}

// array for holding messages to be logged
char debugBuf[8192];
void debugInfo(char*buf)
{
    #if !DEBUG
        return;
    #endif

    long oldSysErr = GetLastError();

    FILE *stream = fopen( "btInfo_debug.txt", "a+" );
    if( stream == NULL )
    {
        MessageBox(NULL, "File 'btInfo_debug.txt' can not be opened!", "", MB_OK);
        return;
    }

    SYSTEMTIME stLocal;
    GetLocalTime(&stLocal);
    fprintf(stream, "%02d/%02d/%d %02d:%02d:%02d.%04d\t",
            stLocal.wDay, stLocal.wMonth, stLocal.wYear,
            stLocal.wHour, stLocal.wMinute, stLocal.wSecond,
            stLocal.wMilliseconds);

    fprintf(stream, buf);
    fclose(stream);

    long newSysErr = GetLastError();

    if (oldSysErr != 0 && oldSysErr != newSysErr)
        SetLastError(oldSysErr);
}
```

Bibliography and Appendix

```
void CloseWinsock()
{
    WSACleanup();
    debugInfo("**INFO* | Winsock 2.2 dll succsesfully unloaded.\n");
}

DWORD OpenWinsock()
{
    // initialize Winsock version 2.2.
    WORD wVersionRequested = wVersionRequested = MAKEWORD( 2, 2 );
    WSADATA wsaData;

    DWORD m_dwError = WSAStartup(wVersionRequested, &wsaData);
    if (m_dwError != 0) return m_dwError; // Unable to initialize Winsock

    debugInfo("**INFO* | Winsock succsesfully loaded.\n");

    // check that we got version 2.2
    if ( LOBYTE( wsaData.wVersion ) != 2 || HIBYTE( wsaData.wVersion ) != 2 )
    {
        CloseWinsock();
        return WSAVERNOTSUPPORTED; // Winsock.dll version out of range
    }

    debugInfo("**INFO* | Winsock 2.2 vesrion succsesfully obtained\n");

    return 0;
}

// returns a list of BT devices in pDevList,
// pItems - keeps the number of actually found devices,
// maxDevices - max number of elements in pDevList array
BTINFO_API DWORD GetBluetoothDevices(BT_DEVICE_INFO *pDevList, long maxDevices, long *pItems)
{
    debugInfo("\n\n*FUNC CALL* | GetBluetoothDevices\n");

    DWORD m_dwError = 0;
    *pItems = 0;

    if ( (m_dwError = OpenWinsock()) != 0)
        return m_dwError;

    UCHAR QueryBuf[2048];
    ULONG wsaBufLen = sizeof(QueryBuf);
    PWSAQUERYSET pWSAQuerySet = (PWSAQUERYSET)QueryBuf;

    INT iResult = 0, iRetryCount = 0;
    ULONG ulFlags = 0;
    HANDLE hLookup = 0;

    int i = 0;

    // looking for a BT devices....
    for (iRetryCount = 0; iRetryCount < CXN_MAX_INQUIRY_RETRY; iRetryCount++)
    {
        ulFlags = LUP_CONTAINERS; // we are looking for devices, not for services...
        ulFlags |= LUP_RETURN_NAME; // I need device name to be returned!
        ulFlags |= LUP_RETURN_ADDR; // and its address too (BTH_ADDR in
        // IpcsaBuffer member of WSAQUERYSET)
        ulFlags |= LUP_RETURN_TYPE; // ask for device type (COD = Class Of Device) in IpServiceClassId
        ulFlags |= LUP_FLUSHCACHE; // reset in order to really scan, not fetch from cache

        iResult = 0;
        hLookup = 0;

        // clean up the buffer...
        ZeroMemory(pWSAQuerySet, wsaBufLen);

        pWSAQuerySet->dwSize = wsaBufLen;
        pWSAQuerySet->dwNameSpace = NS_BTH;
        pWSAQuerySet->IpcsaBuffer = NULL;
    }
}
```

Bibliography and Appendix

```
        debugInfo("**API CALL* | Inquiring devices ... - WSALookupServiceBegin()\n");

// call WSALookupServiceBegin in order to start the lookup
iResult = WSALookupServiceBegin(pWSAQuerySet, ulFlags, &hLookup);

        sprintf(debugBuf, "**API CALL* | after WSALookupServiceBegin(): iResult = %i, hLookup==NULL ?
%i\n", iResult, hLookup == NULL);
        debugInfo(debugBuf);

// got some problem... break the cycle and exit
if ( (iResult != NO_ERROR) || (hLookup == NULL) )
{
        m_dwError = WSAGetLastError();
        printf("=CRITICAL= | WSALookupServiceBegin() failed with error code %d, WSALastError = %d\n", iResult,
WSAGetLastError());
        sprintf(debugBuf, "**CRITICAL* | WSALookupServiceBegin() failed with error code
%d\n", m_dwError);
        debugInfo(debugBuf);
        break;
}

// this loop is to find all BT devices
while ( TRUE )
{
        sprintf(debugBuf, "**CRITICAL* | WSALookupServiceNext() failed with error code
%d\n", m_dwError);
        debugInfo("**API CALL* | Inquiring devices ... - before WSALookupServiceNext()\n");

        if ( NO_ERROR == WSALookupServiceNext(hLookup, ulFlags, &wsaBufLen, pWSAQuerySet) && i <
maxDevices )
        {
                // so we found something!
                debugInfo("**API CALL* | Inquiring devices ... - after
WSALookupServiceNext() - returned no error\n");

                long deviceNameLen = 0;

                if (pWSAQuerySet->lpszServiceInstanceName != NULL)
                {
                        deviceNameLen = (long)strlen(pWSAQuerySet-
>lpszServiceInstanceName);

                        sprintf(debugBuf, "**API CALL* | after WSALookupServiceNext() -
got a device %s (strlen=%i)\n", pWSAQuerySet->lpszServiceInstanceName, deviceNameLen);
                }
                else
                {
                        sprintf(debugBuf, "**API CALL* | after WSALookupServiceNext() -
got a device, but name is NULL!\n");
                }

                pDevList[i].pDeviceName = (LPSTR)CoTaskMemAlloc( deviceNameLen + 1);
                strncpy(pDevList[i].pDeviceName, pWSAQuerySet-
>lpszServiceInstanceName, deviceNameLen);
                pDevList[i].pDeviceName[deviceNameLen] = 0;

                debugInfo(debugBuf);

                DWORD dw = pWSAQuerySet->lpServiceClassId->Data1;
                DWORD mjc = (((dw) & COD_MAJOR_MASK) >> COD_MAJOR_BIT_OFFSET);
                DWORD mnc = (((dw) & COD_MINOR_MASK) >> COD_MINOR_BIT_OFFSET);

                pDevList[i].majorCOD = mjc;
                pDevList[i].majorCOD = mnc;

                PCSADDR_INFO pSockAddr = pWSAQuerySet->lpcsaBuffer;
                SOCKADDR_BTH *pbta = (PSOCKADDR_BTH)pSockAddr-
>RemoteAddr.lSockaddr;

                int w1 = (pbta->btAddr >> 0) & 0xFF,
                w2 = (pbta->btAddr >> 8) & 0xFF,
                w3 = (pbta->btAddr >> 16) & 0xFF,
                w4 = (pbta->btAddr >> 24) & 0xFF,
                w5 = (pbta->btAddr >> 32) & 0xFF,
                w6 = (pbta->btAddr >> 40) & 0xFF;
```

Bibliography and Appendix

```
        sprintf(debugBuf, "\t Remote MAC address: MAC
%02X:%02X:%02X:%02X:%02X:%02X\n", w6, w5, w4, w3, w2, w1);
        debugInfo(debugBuf);

        DWORD addressSize = 20;
        pDevList[i].pMACAddress = (LPSTR)CoTaskMemAlloc(addressSize);
        sprintf(pDevList[i].pMACAddress, "%02X:%02X:%02X:%02X:%02X:%02X\0",
w6, w5, w4, w3, w2, w1);

        debugInfo(pDevList[i].pMACAddress);

        i++;

        debugInfo("**API CALL* | Inquiring devices ... - after
WSALookupServiceNext() - done\n");
    }
    else
    {
        if (WSA_E_NO_MORE == (m_dwError = WSAGetLastError()))
        {
            m_dwError = 0;    // no more devices "error", then clear the
error flag
            debugInfo("**API CALL* | WSALookupServiceNext() returned
WSA_E_NO_MORE (no more devices)\n");
        }
        else if (i > maxDevices)
        {
            m_dwError = 0;
            sprintf(debugBuf, "**CRITICAL* | WSALookupServiceNext() - too
many devices found! return first %i\n", maxDevices);
            debugInfo(debugBuf);
        }
        else
        {
            // and this is "real" error which should be reported to the user
            sprintf(debugBuf, "**CRITICAL* | WSALookupServiceNext() failed
with error code %d\n", m_dwError);
            debugInfo(debugBuf);
        }
        break;
    }
}

        sprintf(debugBuf, "**API CALL* | Stopping lookup - WSALookupServiceEnd(), found %i devices\n",
i);
        debugInfo(debugBuf);

        // stop lookup
        WSALookupServiceEnd(hLookup);
    }

    *pItems = i;    // number of found devices

    sprintf(debugBuf, "**API CALL* | exiting... total found %i devices, error = %i\n", *pItems, m_dwError);
    debugInfo(debugBuf);

    return m_dwError;
}

// makes a nice-looking error string out of windows error code
BTINFO_API void GetErrorString(LPSTR strError, long maxLen, long dwError)
{
    int pos = sprintf(strError, "Error %i: ", dwError);

    if (!FormatMessage(
        //FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dwError, // GetLastError(),
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), // Default language
        strError + pos, // (LPTSTR) &lpMsgBuf,
```

Bibliography and Appendix

```
        maxlen - pos,  
        NULL ))  
    GetErrorString(strError, maxlen, GetLastError());  
    SetLastError(0);  
}
```


Bibliography and Appendix

```
        string sqlStr = "SELECT * FROM " + cliDS.CLIENT_LIST.TableName +
                        " WHERE " +
cliDS.CLIENT_LIST.CLIENT_LIST_IDColumn.ColumnName + "=" + id.ToString();

        SqlHelper.ExecuteDataset(cliDS, Settings.DBConnectionString, CommandType.Text,
sqlStr);

        return cliDS;
    }

    public static ClientsListDS LoadClientByMAC(int id, string mac)
    {
        ClientsListDS cliDS = new ClientsListDS();

        string sqlStr = "SELECT * FROM " + cliDS.CLIENT_LIST.TableName +
                        " WHERE " + cliDS.CLIENT_LIST.MAC_ADDRESSColumn.ColumnName + "=" +
mac + "" +
                        " AND " + cliDS.CLIENT_LIST.CLIENT_LIST_IDColumn.ColumnName + "<>" +
id.ToString();

        SqlHelper.ExecuteDataset(cliDS, Settings.DBConnectionString, CommandType.Text,
sqlStr);

        return cliDS;
    }

    public static void SaveClient(int id, string mac, string surname, string firstName)
    {
        if (id > 0)
            UpdateClient(id, mac, surname, firstName);
        else
            InsertClient(mac, surname, firstName);
    }

    private static void InsertClient(string mac, string surname, string firstName)
    {
        ClientsListDS ds = new ClientsListDS();

        string sqlStr = "INSERT INTO " + ds.CLIENT_LIST.TableName+
                        "(" +
ds.CLIENT_LIST.MAC_ADDRESSColumn.ColumnName + ", " +
ds.CLIENT_LIST.SURNAMEColumn.ColumnName + ", " +
ds.CLIENT_LIST.FIRSTNAMEColumn.ColumnName +
                        ") VALUES " +
                        "(" +
                        "" + mac + "", " +
                        "" + surname + "", " +
                        "" + firstName + "" +
                        ")";

        SqlHelper.ExecuteNonQuery(Settings.DBConnectionString, CommandType.Text,
sqlStr);
    }

    private static void UpdateClient(int id, string mac, string surname, string firstName)
    {
        ClientsListDS ds = new ClientsListDS();

        string sqlStr = "UPDATE " + ds.CLIENT_LIST.TableName + " SET " +
ds.CLIENT_LIST.MAC_ADDRESSColumn.ColumnName + "=" + "" + mac + "", " +
ds.CLIENT_LIST.SURNAMEColumn.ColumnName + "=" + "" + surname + "", " +
ds.CLIENT_LIST.FIRSTNAMEColumn.ColumnName + "=" + "" + firstName + "" +
                        " WHERE " +
ds.CLIENT_LIST.CLIENT_LIST_IDColumn.ColumnName + "=" + id.ToString();

        SqlHelper.ExecuteNonQuery(Settings.DBConnectionString, CommandType.Text,
sqlStr);
    }
}
```

Bibliography and Appendix

```
public static void DeleteClient(int id)
{
    ClientsListDS ds = new ClientsListDS();

    string sqlstr = "DELETE FROM " + ds.CLIENT_LIST.TableName +
                   " WHERE " +
ds.CLIENT_LIST.CLIENT_LIST_IDColumn.ColumnName + "=" + id.ToString();

    SqlHelper.ExecuteNonQuery(Settings.DBConnectionString, CommandType.Text,
sqlstr);
}
}
```