Analysis of Bluetooth Protocol Security

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Todd Baumeister

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2010

Analysis of Bluetooth Protocol Security

By Todd Baumeister

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

_____

Dr. Thomas Gendreau Date
Examination Committee Chairperson

_____

<<Co-adviser's name>> Date
Examination Committee Member

_____

<<Examiner's name>> Date
Examination Committee Member

ABSTRACT

BAUMEISTER, TODD, A., "Analysis of Bluetooth Protocol Security", Master of Software Engineering, 04 2010, (Gendreau, Thomas).

This manuscript describes the analysis of the Bluetooth protocol for known and possible security flaws, and proposes possible solutions to resolve the flaws. The proposed solutions that this manuscript presents are targeted towards the application layer for two reasons. The first is that as of 2008 over two billion Bluetooth devices have been sold. This means that the Bluetooth protocol is already well used, and solutions that require modification of the protocol or hardware will only protect future Bluetooth devices. The second reason the proposed solutions are targeted at the application layer is that not all applications that use Bluetooth require the level of security that is expected in this manuscript. The manuscript also goes on to validate two of the proposed solutions using a proof-of-concept method and using a formal proof.

TABLE OF CONTENTS

Page

LIST OF FIGURES

GLOSSARY

Authorization Server

This is the server that is responsible for maintaining all user and device information for user authentication in the proposed distributed user authentication protocol.

Bluetooth

Short range wireless communication technology.

Bluetooth SIG

The Bluetooth Special Interests Group is responsible for maintaining the Bluetooth protocol specification.

Entropy

In communication theory, Entropy refers to a numerical measure of the uncertainty of an outcome.

HCI

Host Controller Interface is a standardized communication between a host stack and a Bluetooth controller.

HMAC-SHA256

A cryptographic hash function that works with block sizes of 256 bits.

L2CAP

A packet based communication protocol in the Bluetooth stack.

Man-in-the-middle Attack

This is a type of attack where attackers attempt to place themselves in between two communicating devices. This forces all communications to be channeled through them, and so they can control what is sent between the devices.

Nonce

In security engineering, a nonce is an abbreviation of number used once.

RFCOMM

A set of Bluetooth transport protocols that sit on top of the L2CAP layer. RFCOMM provides a reliable data stream that is comparable to TCP.

VERDICT

An assertion that all computer attacks are a combination of one or more of the following improper conditions: validation, exposure, randomness, and deallocation. The resulting taxonomy is the Validation Exposure Randomness Deallocation Improper Conditions Taxonomy (VERDICT).

1. Introduction

The purpose of the research discussed in this manuscript was to find possible solutions to security issues found with the Bluetooth security protocol. All of the solutions proposed were implementable with existing Bluetooth hardware and protocols, and so the solutions were presented at the application layer. This was done for two reasons. The first was that in 2008 over two billion Bluetooth enabled devices had been manufactured, and so Bluetooth was already in heavy use. This means that any solutions that would involve a modification of the Bluetooth protocol or hardware would not fix existing devices. The second reason the proposed solutions are targeted at the application layer was that this manuscript expects the Bluetooth protocol to uphold the highest level of security possible. The problem with that is many Bluetooth devices are battery operated and the computational expense of implementing a high level of security can decrease the battery life of the device. Because of this tradeoff between battery life and level of security, many applications that do not require a high level of security do not implement them.

This research project was split up into three different goals, and the manuscript has its content arranged in the same order. The first goal was to analyze the Bluetooth protocol for any known or possible security flaws. The second goal was to then take those security flaws and find viable solutions that can be implemented at an application layer. The final goal was to prove that a subset of the proposed solutions will effectively solve the issue they were designed for.

2. History of Bluetooth

What is Bluetooth? Before going into the security issues associated with Bluetooth, it first needs to be defined. In addition, the history of Bluetooth will also be covered.

Bluetooth was named after Harald I Bluetooth (Danish Harald Blåtand) the King of Denmark between 940 and 985 AD. King Harald Bluetooth united Denmark and Norway during an era when the region was torn apart by wars and feuding clans. In the way that King Bluetooth united the Scandinavian countries of Europe, wireless Bluetooth is uniting the automotive, mobile phone, and PC industries. Bluetooth is giving each of these industries with their different platforms a common media to communicate with, and thus bringing them together.

Bluetooth is a wireless communication link, operating in the unlicensed ISM band at 2.4 GHz using a frequency hopping transceiver. It allows real-time AV and data communications between Bluetooth enabled hosts. The link protocol is based on time slots [15]. Bluetooth can communicate with other Bluetooth devices at a range of one meter to 100 meters depending on the class of the device. A class 3 device has a maximum range of 1 meter, and a class 1 device has a maximum range of 100 meters. Bluetooth does not require that there is a line of sight between communicating devices; however, line of sight will increase the range that the devices can operate at. The Bluetooth protocol has a built-in pairing mechanism and link level security. Bluetooth is also designed to use low

power consumption, since most of the devices that use Bluetooth will operate entirely on battery power. Bluetooth enabled devices can connect with each other to form a Pico-net. Bluetooth's wireless personal area network (WPAN) technology uses has been standardized by IEEE as 802.15. Examples of some of the common devices that Bluetooth is used in are: Mobile Phones, PCs, PC peripherals, consumer electronics (CE), communications, automotive, and industrial/medical devices [17].

Bluetooth was originally conceived by Ericsson Mobile Communications in 1994 from a study to investigate the feasibility of a low-power low-cost radio interface between mobile phones and their accessories. Bluetooth was released to the public in 1998, under the direction of the Bluetooth Special Interests Group (SIG). Bluetooth SIG was formed by the five companies Ericsson, Nokia, IBM, Toshiba and Intel. By the year 2008 there were more than two billion Bluetooth enabled devices, and the Bluetooth SIG had more than 10,000 members.

3. Bluetooth Protocol Security Background

The first goal of this project was to identify the known and possible security flaws in the Bluetooth protocol. Before each of the security vulnerabilities are discussed in depth, a basic overview of the security features available in the Bluetooth protocol will be discussed.

The Bluetooth specification includes support for several link level security features. Among those features are unidirectional or mutual authentication and encryption. Most of the supported security features are based on a secret link key that is generated during the pairing process, and the key is shared between the connecting devices. The security features of Bluetooth are handled by the Link Manager Protocol (LMP), which can be seen in the complete Bluetooth protocol stack in Figure 1.

Bluetooth can operate in three different security modes. In security mode 1, no security procedures are performed. In security mode 2, service level security is performed. This means that security procedures will be invoked depending on the type of service being used. In this mode security procedures will not be invoked until after a channel is established. Security mode 3 is a link level security. In this mode security is performed in the LMP before a channel is created between devices.

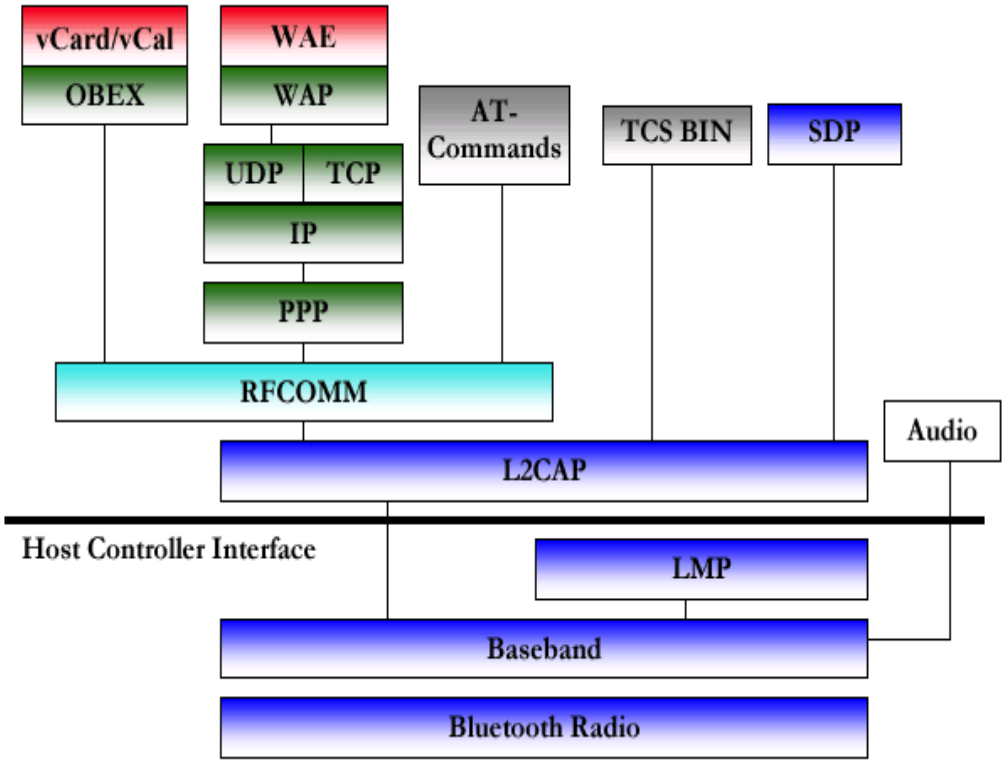


Figure 1: Complete Bluetooth Protocol Stack.

The link level security in Bluetooth performs device level authentication. There are several pieces of information that each device contains and shares with others: Bluetooth device address, user-friendly name, passkey, and Bluetooth device class. Each device is identified by a unique 48-bit Bluetooth device address (BD_ADDR). This device address is similar to the MAC address used in other network communication protocols. Each device also has a user-friendly name that can be up to 248 bytes long. The passkey is used to authenticate two devices that have not previously exchanged data. The final piece of information is the Bluetooth device class, which identifies the type of Bluetooth device and the services supported on the device.

4. Bluetooth Protocol Security Analysis

The Bluetooth protocol flaws discussed in this manuscript were gathered from a variety of other projects and papers. The bulk of the security flaws came from [9]. In this project, the scheme VERDICT was applied to the Bluetooth protocol, which resulted in the discovery of many known issues and a few new ones. The remaining issues were gathered from papers discussing specific security flaws in the Bluetooth protocol [7], [8], [10].

4.1. Man-In-The-Middle 2.0

If there are two legitimate Bluetooth devices A and B that wish to communicate for the first time, the two devices must go through a security process called pairing or bonding. The output of this process is a shared link key that the devices can use to encrypt communications between each other. Bluetooth devices that are running under the 2.0 + EDR (Enhanced Data Rate) specification or prior specifications are susceptible to man-in-the-middle attacks that take place during the pairing process. Since the pin or passkey used during the pairing process is the only source of entropy, it is possible for the attackers to make brute force guesses at the passkey offline with information gathered from the man-in-middle attack.

The fact that there is only one source of entropy makes passive eavesdropping attacks very practical. There are several variations of the man-in-the-middle attacks; however, most of them share the same goal, which is to compromise the communication link between devices. Because of the similarities between the variations on the man-in-the-middle attacks, only one variation of the attack was covered [7].

This passive man-in-the-middle attack exploits a security flaw that is present in the Bluetooth pairing process. In this attack, an attacker will passively listen to communication traffic passing by, and this is illustrated in Figure 2. To understand the security flaw the pairing process is first discussed. The first step of the pairing process is the creation of an initialization key, K_{init}, that is later used to derive the link key. The K_{init} key is computed using the device address of the Bluetooth device (BA), a random number (RN), and a user supplied passkey. In this example, let's make device A the master device and device B the slave. Device A will generate the RN and send it to B as plain text. The BA that is used will depend on the devices pairing together. If one device has a fixed passkey then the other device's BA will be used, but if both devices have a variable passkey then the BA of the slave device will be used. The final piece of information A and B need is the passkey. If a device has an input method then most likely the passkey will be obtained from the user entering it. If the device does not have an input method then the passkey will be fixed by the manufacturer. Now both A and B have all of the required information to generate the K_{init} key. The variables used in the Bluetooth pairing process and how they are constructed can be viewed in Figure 3.

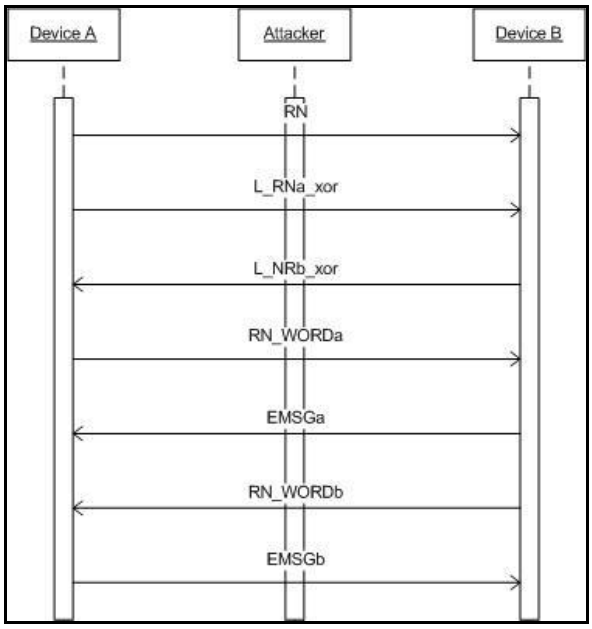


Figure 2: Passive Man-in-the-middle Attack

Now that K_{init} has been calculated, the link key, $K_{AB,}$ can be calculated. A and B will both generate their own random numbers L_RN_A and L_RN_B, respectively. Device A will take L_RN_A and bit-wise XOR it with K_{init} then transmit the result to B. Device B will do the same as A except with L_RN_B. Since both devices know K_{init}, they can take the L_RN XOR K_{init} they receive from the other device and compute the original L_RN by doing another bit-wise XOR with K_{init}. Now that both devices know L_RN_A and L_RN_B, K_{AB} can be calculated.

The final step of the paring process is mutual authentication. Device A will generate a random word RN_WORD, and then transmit it to B. Device B will encrypt RN_WORD using its own BA and K_{AB}. The encrypted message, $EMSG_A$, will be returned to device A where A will compare the returned value to what it originally generated. Once this is completed, the process is repeated again except devices A and B will switch roles.

$K_{init} = f(\ BA_B, RN, passkey)$

$K_{AB} = f(\ L\_RN_A, BA_A, L\_RN_B, BA_B)$

$EMSG_A = f(\ RN\_WORD_A, BA_B, K_{AB})$

$EMSG_B = f(\ RN\_WORD_B, BA_A, K_{AB})$

Figure 3: Bluetooth Pairing Process Variables

If devices A and B have both passed through this entire process without any issues, then both devices now have a shared key K_{AB} that can be used to encrypt any communication between them. The key K_{AB} can be stored in non-volatile memory so that if devices A and B wish to communicate in the future they already have the shared link key and can skip the pairing process.

Now that the pairing process has been discussed, it is possible for a third device T to passively listen to the pairing process of A and B, and then calculate the link key K_{AB}. If T is able to calculate K_{AB} it will then be able to decrypt messages at the link between A and B. When T listens in on the pairing process between A and B it will have the transmitted pairing information in Figure 4 available.

RN - Random number used to generate K_{init}
BA_A - Device A Bluetooth Address
BA_B - Device B Bluetooth Address
$L\_RN_{A\_XOR}$ - XORed with K_{init}
$L\_RN_{B\_XOR}$ - XORed with K_{init}
RN_WORD_A
RN_WORD_B
$EMSG_B$

Figure 4: Publicly Broadcast Bluetooth Pairing Information

With this information available the only unknown value is the passkey. It is possible to make brute force guesses at what the passkey is, and then compare it to the known information. The details of the passkey guessing algorithm or the optimizations that can be performed to make it faster are beyond the scope of this project and therefore are not included in this manuscript. It has been shown that it is possible to crack a 4 digit passkey in ~0.063sec, or a 7 digit passkey in ~76.127sec using a Pentium IV 3Ghz computer [13]. This attack is relatively simple to mount against any two devices that are pairing, but device T would have to be present and listening when A and B are pairing. Pairing between most devices will only ever occur once; however, it has been shown that it is possible to force two devices to rerun the pairing procedure [13].

The man-in-the-middle security vulnerability presents a high level of risk. The attack itself can be performed with relative ease, but would require special hardware because of timing and synchronization issues. Special hardware is not required for the man-in-the-middle attack to be carried out, but it is required to force two devices that are already paired to repair [13]. Since this attack can only be carried out when two devices are pairing, the ability to force repairing allows an attacker to perform this attack at will. This attack can be mounted against any

pair of Bluetooth devices that are pairing using the 2.0 specification or less. The attack will only compromise the integrity of the communication link, and not the Bluetooth devices themselves. Since only the communication link between the devices is compromised the attacker must stay within Bluetooth operating range of the victim devices. The normal operating range of the most powerful class of Bluetooth devices is about 100 meters; however, in some cases the range has been extended to over one mile [5]. This attack can take place without either of the legitimate devices knowing that it happened.

In addition to the passive man-in-the-middle attack outlined here, there is also an active man-in-the-middle attack that is a variation on the above. However, the active attack is much more difficult to mount against two devices. Active man-in-the-middle attackers are described in detail in [12]. Since both of the attacks are very similar, they can be solved using the same counter-measures. If the special hardware can be obtained or built to force the pairing process to rerun between legitimate Bluetooth devices, either type of man-in-the-middle attack could be a very serious security issue. When most users pair Bluetooth devices they will use short passkeys that can be cracked in under a second. After performing a passive man-in-the-middle attack it is possible for the attacker to use the information gathered and use Bluetooth Address Spoofing to assume the identity of either of the two devices that were compromised with this attack. If one or more of the Bluetooth devices communicating does not have input capabilities then secure communications cannot be ensured.

4.2. Man-In-The-Middle 2.1

In the Bluetooth 2.1 + EDR specification, a new pairing process called Secure Simple Pairing (SSP) was introduced. One of the reasons SSP was introduced was to thwart man-in-the-middle attacks against Bluetooth devices. However, SSP is not a full proof pairing process, and it has some weakness that can be exploited.

SSP can operate in four different modes that are dependent on the input/output capabilities of the Bluetooth devices that are pairing. Some of the modes are more secure than others, and so most attacks will focus on the more vulnerable modes. There are several proposed attacks against SSP, but only one of the attacks will be covered here. This section will cover the attack against the passkey entry mode proposed in [12].

The four modes of SSP are as follows. The first mode is called "numeric comparison", and this mode is used when both devices can display a six-digit number. The user is asked to verify that the number displayed is the same on both devices. The second mode is referred to as "just works", and it is used when no IO is available on either of the devices. This mode uses the same protocol as numeric comparison, but no comparison is actually made. The third mode is called "out of band", and it is used when the devices have another channel that can be used to reliably pass information to each other. The final mode is called "passkey entry", and this mode uses a pin or passkey like in the Bluetooth 2.0 specification.

Assume there are two Bluetooth devices A and B that are attempting to pair using SSP. The first step is the capabilities exchange between the A and B. In this step each device will exchange its IO capabilities with the other. The next step is the exchange of public keys, PK. The public keys are generated using Elliptic Curve Diffie-Hellman public-key cryptography [6]. These keys were added to the process to introduce another source of entropy. Next the authentication stage takes place. The purpose of the authentication stage is to verify that the key exchange is

concluded successfully. This is the only step that will vary depending on which mode SSP is operating in. Following the authentication stage, the link key needs to be calculated. Finally, the last step is for A and B to derive encryption keys from the link key.

Assume that there is an attacking device T that was able to get in between A and B and act as an intermediary. The attack described in this section will focus on the capabilities exchange step and the authentication stage with SSP in passkey entry mode. During the capabilities exchange, T can falsify the capabilities of A and B to force the devices to use the passkey entry mode. After the capabilities have been exchanged, A and B will move on to the public key exchange where PK_A and PK_B are exchanged, and then the devices move on to the authentication stage.

In the authentication stage, the passwords or passkeys from each device will be compared one bit at a time. This gives attackers an opportunity to easily brute force the passkey that two pairing devices are using. The attacker can send a bit of the passkey and check if it passed or failed. If sending the bit failed the attacker can just flip that bit and it will be correct the next time they send the bits. The details of how this attack works can be found in [12].

Given the steps in the authentication stage for SSP in passkey entry mode, it is possible to discover the password for a given device in an average of k/2 pairing attempts where k is the length in bits of the password. This attack requires T to attempt to rerun the pairing process with the victim at most k times, and this attack will only work if the password used in each of those attempts does not change. To perform the attack T must listen in on A and B pairing once so it can gain certain pairing information. Now with the pairing information T can attempt

to pair with A or B, and perform this attack to determine the password of the device.

Once T has the password for its victim device, it cannot simple derive the link key like it could in the Bluetooth 2.0 specification because the Diffie-Hellman keys are used as part of the key exchange process. However, now with the password T can directly pair with the victim device.

This is a medium risk security threat. Like the man-in-the-middle 2.0 attack, it can be performed with relative ease but would require special hardware. The effectiveness of the man-in-the-middle attacks has been lowered by limiting the number of situations in which the attack can take place. In the Bluetooth 2.0 + EDR specification all Bluetooth connections were susceptible to passive man-in-the-middle attacks, assuming the passkey was sufficiently short. With the 2.1 specification, man-in-the-middle attacks have only been possible when connecting in certain modes. This version of the Bluetooth specification also effectively eliminates the threat of passive eavesdropping attacks between communicating Bluetooth devices, but it does not fully protect against active man-in-the-middle attacks. The attack described in this section also requires constant rerunning of the pairing process with a victim device, and requires the victim device to consistently use the same password during an attack session. If an active man-in-the-middle attack is successfully carried out, only the communication link between the two victim devices will be compromised.

In conclusion, the newer Bluetooth specification thwarts passive eavesdropping attacks but is still susceptible to active man-in-the-middle attacks. It appears to be a little more difficult to mount man-in-the-middle attacks than it used to be with the older specification, but it is still possible for these attacks to comprise a connection between two valid Bluetooth devices. The risk associated with these

attacks is still significant. If one or more of the communicating Bluetooth devices does not have input capabilities then secure communications cannot be ensured.

4.3. Bluetooth Address Spoofing

Each Bluetooth devices has a unique identifier, a 48-bit Bluetooth Address. This Bluetooth Address is comparable to MAC address used in other networking protocols. When Bluetooth devices connect they are authenticated at the link level, and they use the Bluetooth Address. If a Bluetooth Address is spoofed, it becomes possible for the attacker to assume the identity of another existing Bluetooth device.

Each Bluetooth device should have a static Bluetooth Address; however Cambridge Silicon Radio creates Bluetooth chip-sets that, in combination with specific Bluetooth stacks, will allow the user to dynamically change the Bluetooth Address assigned to a Bluetooth chip-set. Cambridge Silicon Radio is one of the leading Bluetooth chip-set manufactures in the world, and in the 4Q 2004 they held 51% of the market share [16]. The ability to spoof a Bluetooth Address is the basis of a majority of other Bluetooth security issues. Bluetooth devices are identified by their Bluetooth Address, and the ability to spoof that address allows an attacking Bluetooth device to assume the identity of a trusted device.

This is a high risk security threat. Bluetooth Address spoofing is very easy to do. All an attacker would need is Bluetooth hardware that is capable of changing the address and a Bluetooth protocol stack the exposes the functionality to change it. The Bluetooth Address of any device can be spoofed. In the Bluetooth 2.0 +

EDR specification and below, authentication is performed using the Bluetooth Addresses of the devices and a passkey. If legitimate Bluetooth devices A and B are pairing or have already paired and a malicious Bluetooth device T was able to obtain the Bluetooth Address of A and the passkey used during pairing, then T has the ability to assume the identity of A using Bluetooth Address spoofing and force the pairing process to rerun between A and B. When the pairing is rerun T, which is impersonating A, now can jump in and connect with B. As long as B uses the same passkey as when it first connected to A, it is possible for B to pair with T.

Although there may be some legitimate reasons for wanting to spoof a Bluetooth Address, most of the reasons seem to be malicious. Bluetooth Address spoofing is the basis of a majority of the known Bluetooth security issues, and it is not an easy issue to resolve.

4.4. Passkey Usage

A general Bluetooth security threat can be found is the way that passkeys are used when two devices are pairing. As shown in a previously mentioned man-in-the-middle attack, it is a relatively simple and quick process to brute force crack a short passkey. However, in some cases it may not even be necessary to brute force the passkey. In the case of devices that have fixed passkeys, which are unchangeable passkeys set by the manufacturer; it may be possible to look up the passkey in product documentation if the make and model of the device is known.

Although this issue is generally associated with devices that have fixed passkeys, it can also be used on user-generated passkeys. When pairing with a

Bluetooth device that uses user input as a passkey it is always possible you can guess their passkey based off of human behavior, like the passkey "1234" or "0000".

> "*So the combination is... one, two, three, four, five? That's the stupidest combination I've ever heard in my life! The kind of thing an idiot would have on his luggage!*" [4].

The security of fixed passkey devices is maintained by the manufacturers creating the Bluetooth enabled devices. If the manufacturer uses the same passkey for all of the products of a certain make and model, then it is just a case of finding an identical device and looking up the passkey. There is a project a called Car Whisperer [14] that has the sole purpose of exploiting this security issue. Based on the project's findings, an alarmingly high number of devices use the static passkey "0000" or "1234".

This is a medium risk threat. This attack is very easy to mount against a device with a fixed passkey, and can be done with standard Bluetooth equipment. There are several programs available that contain a database of product models to known passkeys. In most cases pairing with a device that has a fixed passkey is not a huge security concern, because many devices that have a fixed passkey also have limited functionality like a mouse, keyboard, or headset. For example, it is possible for attackers to connect to the Bluetooth headset device in car if there is currently no other device paired with it. Then the attacker can use your headset as a listening microphone in your car and eavesdrop on your conversation, or to broadcast messages through it.

In conclusion this attack is mainly geared towards vehicles with Bluetooth headsets in them, but in those cases the cars are almost always moving so the attacker would have to say in range of the vehicle. This attack is easy to carry out, but the rewards of the attacks are far less than other types of attacks.

4.5. Discover Hidden Devices

A general Bluetooth security issue is the ability to discover new devices to attack. Bluetooth devices have the ability to not be seen publicly, but it is still possible to find these devices and get them to respond.

Bluetooth devices have three different modes that they can operate in with respect to device visibility. In silent mode, a Bluetooth device will never accept incoming connections, and it will only listen to Bluetooth traffic. In private or non-discoverable mode, a Bluetooth device will accept incoming connections, but it will not broadcast that it is willing to accept incoming connections. In order to connect to a device in private mode the Bluetooth Address of the device must be known. The final mode is called public or discoverable, and in this mode the Bluetooth device will broadcast that it will accept connections to any device that is within listening range of the device.

The security issue is that when a Bluetooth device is running is private mode it is still possible to discover the Bluetooth Address of that device. The basic principle behind the attack is to ping all possible Bluetooth Addresses and listen to see if a device will respond, and Bluetooth devices that are in private mode will still respond to connection requests. The main problem with this attack is the amount of time it would take to try all possible combination of the 48-bit Bluetooth Address. If the manufacture of the Bluetooth device is known the attack time can be greatly reduced. The Bluetooth SIG will allot distinct ranges of Bluetooth Address to different manufactures to ensure that all Bluetooth devices manufactured will have a unique Bluetooth Address.

This is a low risk security threat. This attack is easy to mount, and it can be done using standard Bluetooth equipment. It is also very effective, and will report any Bluetooth devices in range that are listening in private mode. This attack will not comprise the security of a Bluetooth device by itself; however, once an attacker knows the Bluetooth Address of a device they can use that information in another attack.

This issue is a small threat by itself, but it does open the possibility for devices in private mode to become victims of other Bluetooth attacks.

4.6. Bluetooth Packet Sniffing

If the Bluetooth Address is known for two communicating devices it is possible for a third device to capture the exchanged packets.

When two Bluetooth devices are communicating, the devices will be sending packets of information to each other through radio frequencies. To reduce the chances of passive devices eavesdropping on the communications, a frequency hopping scheme is used to transmit the packets. So a single packet will be sent at one frequency, then the devices will switch frequencies and transmit the next packet. To an outside observer the pattern in which the next frequency is chosen would seem random, but it is actually derived from the Bluetooth Address and clock information on the master device in the communication. Frequency hopping is also used as a mechanism to avoid interference from other devices operating in the same frequency. There are devices that can be purchased that have the capability of analyzing all Bluetooth frequencies and capturing traffic. It is also

possible to take some existing standard Bluetooth chip-sets, and with custom drivers turn them into Bluetooth packet sniffers.

This is a medium threat security risk. This attack requires special equipment to carry out, or specific Bluetooth chip-sets with custom software to control the chip-set. If the equipment can be obtained it is possible to listen to all Bluetooth traffic. The direct damage caused by this attack can vary. The attacker is only going to be able to read packets that are sent in plain-text, and so if Bluetooth is running security mode 1 then the attacker would be able to see all the traffic. However, if the Bluetooth devices are in security mode 3 then the attacker would be able to see the encrypted data but not decrypt it. In security mode 3 the only plain-text messages sent are the pieces of information needed when two devices are pairing. It is possible for an attacker to take the plain-text information gathered when device pairing is executed, and use it as input to another attack.

In conclusion this attack is the basis for many other attacks, but by itself it has limited threat. It will allow that attacker to capture the data sent between two devices, but if the connection is encrypted the attacker will not be able to decipher the content of the data.

4.7. Bluejacking Issue

Bluejacking is a Bluetooth security issue that allows a Bluetooth device to send an anonymous business card to another Bluetooth device. The business card is sent to the Bluetooth device using the Object Exchange protocol in Bluetooth. A business card is an electronic data form that contains contact information of individuals and businesses.

This issue is caused by a loophole in the Bluetooth messaging system in certain implementations of the Bluetooth specification. An attacking Bluetooth device will generate a business card, and then send this business card to the victim Bluetooth device. The attackers will usually put a message in the name field that will display on the victims Bluetooth device. The victim device will be asked to accept the new business card.

This is a low threat security risk. There are publicly available software packages that can be installed on cell phones to carry out this attack. There are only certain models of cell phones that are susceptible to this attack, and there has been updates released for most of those phones. This attack does not in any way alter or delete existing information on the victim's Bluetooth device. The victim device can decline the request to add the new business card to their contacts, and even if they do there is minimal risk in accepting. This kind of attack is not targeted towards the Bluetooth device, but at the device's user.

In conclusion this is a minor threat and has mostly been dealt with. The only devices that would still be susceptible to this attack would be older cell phones that have not installed updates.

4.8. Bluebugging and Bluesnarfing

The next two issues, Bluebugging and Bluesnarfing, are specific security issues that were an issue with older Bluetooth enabled cell phones. Bluebugging allowed an attacker to gain control of a victim's phone and make calls, send text messages, and read and write phone contacts. Bluesnarfing allowed an attacker to access the victim's phone book and calendar. Both of these attacks can be carried out without the knowledge of the victim.

Both of these issues are caused by improper implementations of the Bluetooth specification. The improper implementations allow an attacking Bluetooth to connect on a covert channel on the victim device and issue AT commands.

This is a medium risk security threat. There are publicly available software packages that can be installed on cell phones to carry out both of these attacks. There are only certain phone manufactures that were susceptible to attack. Most of the phones that had this issue have had updated released to fix this issue, and so the only vulnerable phones would be ones that have not installed updates. If a phone was compromised using either of these attacks it is possible for the attacker to have full access including delete permissions to the victim's contacts and calendar. With Bluebugging the attacker would also be able to place calls and text messages from the victim phone. This could cause financial implications for the victim if phone services are used that get charged to a user's phone bill.

In conclusion this was a very serious threat at one time for certain phone models that had Bluetooth abilities; however, the issue has been almost entirely resolved except for a few lingering devices that would need to be patched with the latest software update.

4.9. DoS

Bluetooth devices can be attacked with Denial of Service attacks, which will prevent the victim Bluetooth device from receiving incoming Bluetooth transmissions. It is possible for a single Bluetooth device to simultaneously attack up to three different Bluetooth devices within range.

This attack is caused by an attacking Bluetooth device continuously sending connection requests to a victim Bluetooth device. This will cause the victim device to have to continuously process the bogus connection requests and not be able to process valid connection requests. There has been instances where a DoS attack can actually incapacitate the device it is attacking. There are certain phone models that will pop up a modal dialog that will ask if the user wishes to accept the incoming dialog. In the time it takes the user to say yes or no the next connection request is in and the dialog will immediately pop up again.

This is a low risk security issue. This attack is easy to carry out and can be done using standard Bluetooth equipment. Any Bluetooth device is susceptible to this attack. However, Bluetooth piconets are not designed for central computing, and so attacking a single Bluetooth device will only affect a very small number of other Bluetooth devices, as opposed to attacking a web server which could affect millions of users. On a majority of Bluetooth devices, the Bluetooth radio can be turned off during a DoS attack. If this attack is performed on battery-operated devices, it is possible that this attack will shorten the battery life of the device with the extra Bluetooth radio transmissions.

This attack is very simple to carry out, but it has very small rewards for the attacker. However, it is possible to use this attack in conjunction with an active

man-in-the-middle attack to ensure the middle attacking device has all traffic routed through it.

5. Proposed Solutions

The second goal of this project was to take the discovered security flaws and find viable solutions that can be implemented at an application layer. Not all of the attacks presented in this manuscript have a solution that can be implemented at the application layer due to the nature of the attack. However, most of the attacks do have application level counter-measures.

5.1. Man-In-The-Middle 2.0

The main vulnerability in this attack is the passkey used to generate the link key. The length of the passkey greatly influences the time it would take to guess the passkey, or if it is even possible for the passkey to be guessed. If the passkey is 64 bits (19 digits) or greater the algorithm used to guess the passkey will begin returning false positives to the attacker. The length of the passkey used can be forced to meet certain requirement at the application layer between two devices that both have user input capabilities; however, this cannot be enforced when one of the devices has a fixed passkey, and in this case the passkey length is usually only 4 digits. In most cases where custom applications are communicating through Bluetooth, both of the devices will have user input capabilities. The only issue with this resolution is that if the two Bluetooth devices have already paired before the custom applications start communicating, the custom application would

have to terminate the connection and force the users to run the pairing process again to enforce the passkey length.

If enforcing a minimum passkey length is not possible, or if the link level encryption Bluetooth offers does not provide sufficient security, then it would be beneficial to implement encryption at the application layer. In this case, symmetric encryption would probably be preferred especially in situations where device battery power is an issue. The shared keys can be distributed by using the Diffie-Hellman key exchange, instead of the less secure key distribution method used by Bluetooth link level security. This way, even if the Bluetooth link level security is compromised, it is still possible to maintain the integrity of the communications between the custom applications. This solution would also protect the application's communications if the Bluetooth link were to have its security mode demoted.

Another solution to this issue is to use a Bluetooth device that supports the 2.1 + EDR specification. This security issue is addressed in this version of the specification; however, it does introduce some new security issues that are discussed later in this manuscript.

None of these solutions up to this point will protect the Bluetooth communications against active man-in-the-middle attacks. It is much more difficult for an attacker to mount an active man-in-the-middle attack than a passive man-in-the-middle attack, and special hardware would be required. In an active man-in-the-middle attack all communications between A and B would go through T, and T would pretend to be device B when communicating with A and would also pretend to be device A when communicating with B. This allows T to decipher all traffic between the devices. To protect against this both A and B would require input capabilities. Then encryption can be implemented at the

application layer using symmetric encryption for communications between A and B, and asymmetric encryption can be used to exchange the symmetric key. When the symmetric key is being exchanged, a passkey that is user supplied can also be exchanged with the asymmetric encryption as a means of authentication. The passkey used here has the same purpose as the passkey required in the pairing process; however the passkey is now being exchanged in a more secure manner. This method would require a user to enter a passkey twice if the two devices are pairing for the first time. This is the preferred way to secure the communications between two Bluetooth devices and prevent man-in-the-middle attacks from compromising the communication link.

5.2. Man-In-The-Middle 2.1

At the application level there are several things that can be done to avoid compromising the Bluetooth connection. For example, after a connection has been established between Bluetooth devices using SSP, the connection could have a property that specifies if the connection is authenticated. For a connection to be authenticated only certain SSP connection modes can be used that help guarantee the connection has not been tampered with. At the application layer you can refuse to use a Bluetooth connection unless it is authenticated.

Implementing encryption at the application layer would be another solution, but this can only be done if both devices have IO capabilities. The proposed recommendation uses a symmetric key encryption between the devices to reduce power consumption, and exchanges the keys using the Diffie-Hellman key Exchange. A passkey would have to also be used at the application layer to provide authentication during the Diffie-Hellman key exchange. This passkey would be encrypted with a public key and compared by each device to see if it

matches their passkey. With this method, it is still possible to maintain the integrity of the communications between the applications even when the integrity of the Bluetooth connection is compromised at the link layer. This solution is the same as the numeric comparison mode of SSP except it would also protect the application's communications if the Bluetooth link were to have its security mode demoted. This solution was proposed for the man-in-the-middle 2.0 attack, but it will also be effective in this version of the man-in-the-middle attack.

One of the solutions to the man-in-the-middle attacks for the Bluetooth 2.0 + EDR specification was to increase the length of the passkey used when pairing. This is not as effective in Bluetooth 2.1 + EDR specification because this attack only takes k/2 pairing attempts to crack the passkey.

5.3. Bluetooth Address Spoofing

To resolve this issue at the application layer, user level authentication can be used. Using two-level authentication means security does not rely solely on "what they have", in this case the Bluetooth Address, but also "what they know". One of the issues with implementing this kind of security counter measure is that Bluetooth networks are not centralized, and so there is no way of setting up a central server to perform authentication. User authentication would have to be performed on a per device basis, and there would need to be some way to synchronize user information changes across all devices. There are other ways to resolve this issue, but they all require that an additional piece of security information to be added to the equation.

5.4. Passkey Usage

Most devices that use fixed passkeys have them because there is no user input available. Because of this it is unlikely that these devices will have the ability to run custom applications on them. One protection against this is to buy Bluetooth devices where the manufacturer has taken proper precautions when choosing passkeys for their devices. If a custom application is running on a device with a fixed passkey, it may be necessary to add another method of authentication at the application layer. One example would be to add user-based authentication to the application. The only issue with this, as mentioned before, is that there would need to be a mechanism to maintain and update the user information in a distributed network environment.

5.5. Discover Hidden Devices

A solution to this issue is to turn a Bluetooth device to silent mode or off when it is not in use. There are no solutions to this issue that would be implementable in the application layer.

5.6. Bluetooth Packet Sniffing

A solution at the application layer to ensure that packet sniffing would not compromise the security of an application would be to encrypt all traffic from application to application. That way if a Bluetooth device were to have its security

mode turned down to level 1, the communications between your applications would still be safe.

5.7. Bluejacking

This is a known issue so most companies have included fixes in their Bluetooth Object exchange protocols to resolve this issue. Users simply need to make sure all device updates have been installed. Future devices should include an update mechanism that would allow patches to be sent out to the users when a security flaw is revealed. It will not prevent security threats from happening, but it will allow them to be managed when they are discovered.

5.8. Bluebugging and Bluesnarfing

This issue was resolved by the phone manufacturers as a patch to their Bluetooth protocol stacks. If using a device that was targeted by this attack, users should make sure to install the latest updates for the device. However, it is possible for another security flaw to be discovered that compromises full access to the device like this attack. To remedy this, custom applications that store persistent information on a Bluetooth device should encrypt the data so that if the device's file system is compromised the data will still be safe. A guideline to do this is the encrypted storage design pattern [11]. The design pattern is geared towards server architecture, but can still be applied to this situation.

5.9. DoS

There are two solutions to this issue. The first would be to simply take the victim device out of range of the attacking device. If the attacker is using standard Bluetooth equipment then that would require moving a little over ten meters away from the attacker. The second solution is to turn off the Bluetooth radio if the attacker is using the attack to drain the devices battery life.

If a custom application requires Bluetooth communications availability as part of its normal operations, fail over systems may need to be put into place in case the Bluetooth communication is temporary unavailable. The fail over system used would depend on the application domain, but it should be assumed that the network connection will not be available 100% of the time.

6. Proposed Solution Verification

The third and final goal of this project was to prove that a subset of the proposed solutions will effectively solve the issues they were designed for. To do this, two solutions will be selected from the list of proposed solutions. The first solution will have a proof-of-concept developed to show that it will in fact work in at least one given situation. The second solution will have a formal proof run against to try and verify that it can be used for all given situations.

6.1 Proof-of-Concept

The solution implemented in the proof-of-concept was adding application level encryption to verify that it can protect against man-in-the-middle attacks. A secure key exchange was invoked so that a symmetric key can be used to encrypt messages at the application layer. The symmetric key would be exchanged using RSA key encryption along with a user supplied passkey that would be used for authentication purposes. This solution was proposed as a fix for several discovered Bluetooth protocol flaws; however only one of the flaws will be discussed in this proof-of-concept.

The Bluetooth protocol flaw being addressed in this proof-of-concept is an issue with the Bluetooth key exchange protocol. If two communicating Bluetooth devices are required to communicate in secure mode, then the two devices must first execute a pairing process that establishes a secret-shared key (link key) that

can be used encrypt future communications. There are several variations of how the pairing process can executed, but the most prominent version requires a pin code to be entered on both Bluetooth devices, and so the pins can be compared. The problem is that this pin code is the only source of entropy with this pairing process, and given enough information a malicious third party can crack the pin code that is being used. Once the pin code has been cracked, then the link key that the two devices are using can be calculated. With the link key, it is possible to calculate the encryption keys that two devices are using, and then decipher their communications.

It is not possible to fix this security issue directly, as it is a problem with the protocol design, but it can be resolved at the application layer. This means that users will not be able to prevent this attack from compromising the communication link between two devices, but they will be able to keep the application data link secure even though the communication link it is traveling on is not secure. The solution proposed will be implemented to secure the data being sent between applications. It is still possible to crack the encryption on the data; however, the time it would take to crack this encryption is estimated to be terms of years, as opposed to the seconds that it takes to exploit the design flaw in the Bluetooth protocol and crack the pin code. The modified Bluetooth protocol stack can be seen in Figure 5. The RSA security layer represents this proposed solution, and it sits in between the RFCOMM layer and the application layer. The RSA security layer is responsible for any data encryption before the data is given to the RFCOMM layer. A detailed discussion of the development of proof-of-concept software can be found in appendix B.

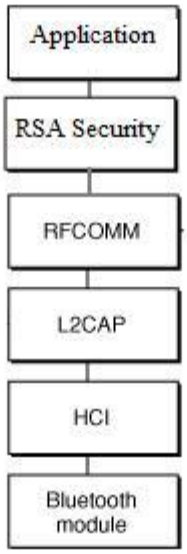


Figure 5: Protocol Stack with RSA Encryption

In the proof-of-concept, an environment was created that mimics the attack process, where a third, malicious party is able to obtain the link key that is being used between two legitimate Bluetooth devices. With this key, the attacker would then be able to decrypt the messages sent between the two devices without the devices knowing. The proposed solution was then implemented, and the attack process was run again.

The project results were successful. An environment was constructed were an attacker was able to perform a passive man-in-the-middle attack, and successfully obtain enough information from the pairing process to brute force the passkey offline. With this information, the attacker was able to compromise the link level communications between the Bluetooth devices. Once the proposed solution was put into place, the attacker was still able to compromise communications at a link level. However, since the proposed solution ran at the application level, the attacker was unable to decipher the data being sent between the applications.

In conclusion the proposed solution prevented the attacker from gaining access to the decrypted information being sent between the two devices; however, the solution only protected the communications between the applications that implemented the proposed solutions and not all of the communications between the two devices. This solution still has merit in real world applications where a level of security that is greater than what available in the Bluetooth protocol is needed. However, this proof-of-concept only proves that the proposed solution will be effective against passive man-in-the-middle attacks that compromise the integrity of a communication link by listening to the pairing process and cracking the pin code used. There are other variations on the man-in-the-middle attack, and different modes of device pairing that can be carried out. To prove that this solution works for those situations additional proof-of-concepts would need to be developed.

A portion of the proof-of-concept project required that an environment be created where an attacker would be able to perform a passive man-in-middle attack. The capturing of all communications between two Bluetooth devices can be done in one of two ways. First, a piece of hardware called a Bluetooth sniffer can be used to intercept communications between two devices. Second, there are Bluetooth protocol stacks that allow the communications between that device and another to be captured. The second method was used in the proof-of-concept project because of the level of reliability it produced. However, the first method was explored.

The cost to buy a professional manufactured Bluetooth sniffer is approximately eight thousand dollars. There are cheaper alternatives to this, but the cost still ranges in the thousands of dollars. In the process of researching, specifications to build a Bluetooth sniffer were found that use consumer available hardware and readily available software at the cost of approximately thirty dollars. The

Bluetooth sniffer was successfully constructed; however it produced inconsistent results. If further time had been invested in this Bluetooth sniffer it is possible that a reliable, low cost sniffer could be constructed. A detailed discussion of the development of the Bluetooth sniffer can be found in appendix A.

In conclusion a Bluetooth sniffer can be used to exploit many Bluetooth security flaws, and it is possible for anyone to build a low cost Bluetooth sniffer. In this project the results of the Bluetooth sniffer were inconsistent; however, it did produce valid results on occasion. If different combination of hardware and software were used it is possible that a reliable Bluetooth sniffer could be built. This means that anyone could listen to Bluetooth communications between two devices for the cost of a single Bluetooth radio chipset.

6.2 Formal Proof

Several of the Bluetooth security flaws discussed in this manuscript had a user authentication system as their proposed solution. The issue with this type of authentication is that Bluetooth does not form a centralized network, and so it is difficult to provide a single server that can provide user authentication and management. To remedy this issue a novel distributed user authentication protocol was proposed, and formal analysis was performed to verify the protocol. The protocol modeled here is not the first version; a protocol was designed, analyzed with the formal proof tool, and corrected. This iteration was performed several times until a secure protocol was established.

The proposed Bluetooth user authentication protocol has two different node types in it. There is an authorization server that is responsible for housing the user information, modifying user information, and serving as a certificate authority for

any devices that choose to authenticate though it. The other type of node is an end device node that can take on one or both of the following roles. The first role is “request service”, and in this role the node will be requesting services on other end devices. The other role is “receive requests”, and in this mode the end device will be processing requests it receives for services it hosts.

The authorization server is the central point of security in this protocol. This means that this server will contain the most current list of users and end devices in the system, and in addition will also be acting as a certificate authority for the end devices. This security protocol is considered to be distributed because actual user authentication is not performed at the authorization server, but is instead performed at the end devices. When an end device connects to an authorization server, the end device will obtain the latest version of a user information database. This user information database contains a list of all the users in the given authorization network. Part of that user information is a user's secret hashed with an authorization server secret. This means that if another end device wishes to authenticate a user they need to provide the user, user's secret, and the authorization server’s secret to the end device. Since the user information is distributed to the end devices, the end devices are not required to have a constant connection the authorization server to perform user authorizations.

Since the user information will be distributed to the end devices registered with the authentication server, it is necessary to place expiration dates on the user information to force end device to periodically update with the authentication server, and to prevent attacks from being carried out that use old user information.

Assume that S stands for a given authorization server, and that A and B are end devices that are registered with the authorization server S. Given this information the proposed distributed user authentication protocol can be modeled. This is an

idealized model of the protocol so the security of the protocol can be analyzed, which means that trivial steps that do not affect the security of the protocol will be omitted. It is also assumed that before this protocol is executed all of the end devices registered with a given authorization server have a private and public key pair, and that any devices registered with an authorization server have a means to obtain the public keys of any other device registered with that authorization server. The model of the steps carried out in the protocol can be found in Figure 6.

1. A -> S : { N_A, A }_pkS
2. S -> A : { N_A, m }_pkA, { N_A }_skS
3. B -> S : { N_B, B, U }_pkS
4. S -> B : { N_B, _sSU}_pkB, { N_B }_skS
5. B -> A : { N_C, B, U, _sSU }_pkA
6. A -> B : { N_C, _sAB }_pkB, { N_C }_skA

Figure 6: Distributed User Authentication Protocol

The format of the notation used in Figure 6 is message path, followed by a colon, and then the content of the message sent. So step one would read, device A is sending a message to device S that contains {N_A, A}_pkS. In the section of the notation that contains the message content two common symbols can be seen { }_pkX and { }_skX. The former means that anything in between the brackets will be encrypted with the public key of X, and the later means it will be encrypted with the private key of X. Commas are used to indicate concatenation. The message content of step one would read; nonce A and host A are encrypted with the public key of host S.

This protocol models all of the steps that would be needed for an end device B to request the use of a service on end device A. Steps one through four are only executed when the end devices first register with the authorization server, or new

security information has been created and the end device must update with the authorization server. Steps five and six actually perform the user authorization between two devices. Since the information from the previous steps can be stored on the end devices, steps five and six can be executed many times and the previous steps only need to be executed occasionally.

In step one the end device A is sending a request to the authorization server to obtain the latest version of the user information database. This request consists of the address of A and a nonce N_A encrypted with the public key of S. When S receives this request it will verify that the nonce has not been used recently, and that A is a registered end device.

In step two, S will respond to A with the nonce N_A and the user information database encrypted with the public key of A. It will also send the nonce N_A hashed with the private key of S. When A receives this it will need to verify that both nonce values match N_A. The second hashed nonce is to verify that the message came from S. Steps one and two have established a user information database on an end device.

In step three, B is requesting the user's authorization server secret from the authorization server by sending a nonce N_B, address of B, and user authentication information U.

In step four, the authorization server will send the nonce N_B and the user's authorization server secret encrypted with B's public key. The hash of N_B will also be sent to verify that message originated from S. Steps three and four have distributed the user's authorization server secret, which is needed by any end device that wishes to authenticate a user with another end device. The secret proves that the end device at some point contacted the authorization server, and

the end device is registered with the authorization server. Each user registered on the authorization server will have their own authorization server secret, _sSU.

In step five, B is requesting to use a service on device A. B is sending a nonce N_C, address of B, user's secret information, and the user's authorization server secret all encrypted with the public key of A. Once A receives this it will check if the user's information and the user's authorization server secret hashed together equal what is stored in the user information database.

In step six, A is sending the nonce N_C and a symmetric key that A and B can use to communicate encrypted with the public key of B. In addition the hash of N_C is being sent to verify that the message originated from A. With the symmetric key, A and B will be able to communicate for a limited period of time until the key times out or is revoked by one of the end devices.

To verify the correctness of this protocol a formal proof tool called ProVerif [3], [1] was used. This tool takes a protocol modeled in PI calculus and attempts to prove certain security properties about the protocol. Running this tool proved that the proposed protocol would allow end devices to authenticate with each other without an attacker being able to obtain the user information database, the user's authorization server secret, or the symmetric key shared between A and B.

The files that ProVerif used as input to prove the correctness of this protocol can be found in the appendix under the sections C.1 and C.2. The proof of the protocol was split up into two separate proofs to simplify the input files into ProVerif. Steps 1-2 in Figure 6 are verified in appendix section C.1, and steps 3-6 are verified in appendix section C.2. In the following discussion, the PI calculus used to prove the protocols correctness will be mapped back to the steps of the protocol. The PI calculus used in ProVerif will be represented in *italics*.

Additional information on the variation of PI calculus that ProVerif uses can be found in [1] and [3].

Step one in Figure 6 is modeled in PI calculus with the following entries in the ProVerif input file. In process A, the statements *new Na; out(c, encrypt((Na, hostA), pkS));* model the creation of a nonce Na and then sending out that nonce encrypted with the public key of host S. Process S contains the statements *in(c,m); let (NY, hostY) = decrypt(m, skS)*. These statements are responsible for capturing the step 1 sent from host A. Once process S gets the message *m* it will decrypt it with the private key of S, and then store the nonce and host passed in. This information will then be used in later steps.

Step two is modeled with the following statements. In process S, the statement *out(c,(encrypt((NY,mSADB),pkA), sign(NY, skS)))* is sending out the nonce stored in *NY* and the user information database encrypted with the public key of host A concatenated with the signature of the nonce *NY*. Process A has the following statements to capture and verify this message; *in(c, (ms, hs)); let (=Na,mASDB) = decrypt(ms,skA) in let =Na = checksign(hs, pkS) in*. These statements will decrypt the message *ms* using the private key of host A, and verify the nonce in the message matches the nonce *Na* that it generated in step one. These statements will also verify that the message *hs* is the signature of the nonce *Na*.

Step three is modeled with the following statements. In process B, the statements *new Nb; out(c, encrypt((Nb, hostB, userU), pkS));* are generating a new nonce *Nb* and then sending that nonce, host B, and the user information all encrypted with the public key of host S. Process S contains the statements *in(c,m); let (NY, hostY, userY) = decrypt(m, skS) in* to capture the message sent from process B. These statements are decrypting the message *m* and storing the

nonce in *NY*, the host in *hostY*, and the user information in *userY*. This information will be used in later steps.

Step four is modeled with the following statements. In process S the statements *out(c,(encrypt((NY,sSU),pkB), sign(NY, skS)))* will encrypt the nonce stored in *NY* and the user's authorization server secret with the public key of host B, and send that message concatenated with the signature of the nonce *NY*. Process B contains the statements *in(c, (ms, hs)); let ((=Nb,=sSU), =Nb) = (decrypt(ms,skB), checksign(hs, pkS)) in*, which are responsible for capturing the message sent in step four. These statements will decrypt the message *ms* with private key of host B, and verify that the nonce matches *Nb*. It will also verify the *hs* is the signature of the nonce *Nb*.

Step five is modeled with the following statements. In the process B the statements *new Nc; out(c, encrypt((Nc, hostB, userU, sSU),pkA));* will encrypt a new nonce *Nc*, the host in *hostB*, the user information in *userU*, and the user's authorization server secret with the public key of host A. Process A contains the statements *in(c, m2); let (Nz, hostZ, userZ, sSZ) = decrypt(m2, skA)* will capture the message sent in step five. These statements will decrypt the message *m2* with host A's private key and store the nonce in *Nz*, the host in *hostZ*, the user information in *userZ*, and the user's authorization server secret in *sSZ*.

Step six is the final step is modeled with the following statements. In process A the statements *out(c, (encrypt((Nz,sAB, hostA),pkB), sign(Nz, skA)))* will encrypt the nonce *Nz*, the secret key between host A and host B, and host A with the public key of host B. It will also concatenate the signature of *Nz*. Process B has the statements *in(c, (ms2, hs2))* which will capture the message sent in step six.

Although the formal proof tool was able to verify certain properties about the protocol, there were some properties that couldn't be modeled with the tool. The tool was able to verify the exchange of information was handled securely; however, the protocol uses a hash of a user secret with the user's authorization server secret to verify a user's identity. There was no way of modeling this behavior in the tool.

In conclusion, this protocol is capable of distributing the information needed to end devices to authenticate with each other securely. There will need to be further discovery done on this protocol to verify the impacts of user information databases being compromised on the end devices. The formal proof techniques used here are very time consuming; however, they do prove very concrete benefits over the proof-of-concept. The time required to perform a formal analysis can be drastically cut down by the use of a formal proof tool that will automate much of the proof process, but the use of formal proof tools also require a steep learning curve.

7. Continuing Work

If our society continues on its current trend, more and more devices will become wireless. As the number of wireless devices increases, the availability of a central network topology may not always be available, and a distributed user authentication protocol will be needed for devices to authenticate with each other. Further work can be done in developing a distributed user authentication protocol. There are several other protocol security models that can be used to verify a protocol such as the probability model. If the distributed user authentication protocol were to become viable it will also need to have capabilities to allow users from an authorization server to authenticate themselves with a different authorization server where those do not initially exist.

Bibliography

1. Abadi, M. Blanchet, B. Fournet, C., “Just Fast Keying in the Pi Calculus”, *ACM Transactions on Information and System Security (TISSEC)*, 10(3):1-59, July 2007.

2. Bandyopadhyay, S. Majumdar, A. Ghosh, O. Chatterjee, S. Chattopadhyay, S., “A proposal for improvement in service-level security architecture of Bluetooth”, *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region* Volume 3, 15-17 Oct. 2003, pp. 1058-1061.

3. B. Blanchet, J., " ProVerif: Cryptographic protocol verifier in the formal model," *proverif.ens.fr*. [Online]. Available: http://www.proverif.ens.fr. [Accessed: April. 1, 2010].

4. Spaceballs. Dir. Mel Brooks. 1987. DVD. M.G.M., 2005.

5. Cheung, H., "How To: Building a BlueSniper Rifle – Part 1," *SmallNet-Builder*, Pudai LLC, homepage, 2005. [Online]. Available: http://www.smallnetbuilder.com/content/view/24256/98. [Accessed: June 20 2009].

6. Barker, Elaine. Johnson, Don. Smid, Miles., “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography”, *NIST Special Publication 800-56A*. March 2007.

7. Haataja, K.M.J.; Hypponen, K., “Man-In-The-Middle Attacks on Bluetooth: a comparative analysis, a novel attack, and countermeasures”, *Communications, Control and Signal Processing, 2008*. ISCCSP 2008. 3rd International Symposium, 12-14 March 2008, pp. 1096-1102.

8. Haataja, K. Toivanen, P., “Practical Man-in-the-Middle Attacks Against Bluetooth Secure Simple Pairing”, *Wireless Communications, Networking and Mobile Computing, 2008*. WiCOM '08. 4th International Conference, 12-14 Oct. 2008, pp. 1–5.

9. Hager, C. Midkiff, S., “An analysis of Bluetooth security vulnerabilities”, *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE* Volume 3, 20-20 March 2003. pp. 1825-1831.

10. Hager, C. Midkiff, S., “Demonstrating vulnerabilities in Bluetooth security”, Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE Volume 3, 1-5 Dec. 2003, pp. 1420-1424.

11. Kienzle, D. Elder, M. Tyree, D. Edwards-Hewitt, J. , "Security Patterns Repository Version 1.0," November 4, 2003. [Online]. Available: http://www.scrypt.net/~celer/securitypatterns/repository.pdf. [Accessed: July 20, 2009].

12. Lindell, A., "Attacks on the Pairing Protocol of Bluetooth v2.1," *Black Hat*, June 25, 2008. [Online]. Available: http://www.blackhat.com/presentations/bh-

usa-08/Lindell/BH_US_08_Lindell_Bluetooth_2.1_New_Vulnerabilities.pdf. [Accessed: July 10, 2009].

13. Shaked, Y. Wool, A., "Cracking the Bluetooth PIN," *Tel Aviv University*, February 5, 2005. [Online]. Available: http://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/. [Accessed: July 1, 2009].

14. "Car Whisperer," *Trifinite*, February 29, 2008. [Online]. Available: http://trifinite.org/trifinite_stuff_carwhisperer.html. [Accessed: July 20, 2009].

15. "Core Specification v2.0 + EDR," *bluetooth.com*. [Online]. Available: http://www.bluetooth.com/German/Technology/Building/Pages/Specifcation.aspx .[Accessed: April. 1, 2010].

16. "CSR plc acquisition of Clarity Technologies, Inc.," *Cambridge Silicon Radio*, March 14, 2005. [Online]. Available: http://www.csr.com/pr/pr190.htm. [Accessed: August 1, 2009].

17. "History of Bluetooth Technology," *bluetooth.com*. [Online]. Available: http://www.bluetooth.com/English/SIG/Pages/History_of_the_SIG.aspx. [Accessed: April. 1, 2010].

18. *backtrack-linux.org*. [Online]. Available: http://www.backtrack-linux.org/. [Accessed: May 6, 2010].

Appendix A: Development of A Bluetooth Sniffer

One of the requirements of the proof-of-concept portion of this project was to create an environment where an attacker could capture the data being sent between two Bluetooth devices, and so the proposed solution to the security issue could be tested. The details of the proof-of-concept can be found in appendix B. To capture the communications between two Bluetooth devices, a lost cost Bluetooth sniffer using consumer available parts was developed. The Bluetooth sniffer cost about thirty dollars, which compared to commercially sold sniffers which sell for thousands of dollars, was inexpensive.

The hardware required to build the Bluetooth sniffer was a specific model of a Bluetooth chipset manufactured by Cambridge Silicon Radio. Once the hardware was acquired, the firmware running on the chipset had to be replaced. Many of the tools used to change the device properties and firmware on the Bluetooth chipset came from a Linux distribution called Back Track [18]. Back Track focuses on identifying and exploiting security vulnerabilities by collecting various security related tools into a single environment where they can be run from.

The final piece required to run the Bluetooth sniffer was software to interpret the packets sent from the updated hardware. Back Track contained some simple source code for interpreting the data packets; however, it needed to be extended to capture the specific data being exchanged during a Bluetooth pairing process.

A low cost Bluetooth sniffer was created; however, because of time constraints in the project the sniffer was not completed. The sniffer was operational, and it could capture the data being sent between two pairing Bluetooth device; however, it was not very reliable. It often took several attempts before the sniffer was able to capture any data being sent between two Bluetooth devices. The sniffer also required to be synchronized with the clock of the two Bluetooth devices it was eavesdropping on, and this needed to be performed on time intervals of approximately one minute.

Appendix B: Development of Application Level Encryption protocol

B.1. Software Development

The purpose of this portion of the project was to create some software that can be used as a proof of concept for a proposed solution to a known Bluetooth protocol security issue. The known issue with the Bluetooth protocol is that if the pairing process between two devices can be observed by a third party, then that party can use the information shared between the pairing devices to derive the pairing pin that was used. Given a sufficiently short pairing key is used by the devices pairing; the third party can derive the pairing key used in a matter of a few seconds. Once the pairing key is known by a third party, they can then calculate the encryption key used for the communications between the two devices. This project can be broken up into two different sections. The first section (software package: Bluetooth Communicator) will consist of some software that utilizes the Bluetooth communication protocol to send information to another instance of its self running on another computer. This software will act as the two legitimate parties communicating information with each other. The second section (software package: Bluetooth Interceptor) will be a piece of software that intercepts the communications between a pair of legitimate devices that are communicating, and then will attempt to decipher the messages going between the legitimate devices.

Software Package: Bluetooth Communicator

The software that will be used to emulate two legitimate devices communication will be discussed first. This software is required to be able to be opened on two different computers that both have Bluetooth communication capabilities. The one of the two instances will then start hosting a server service that the other instance will be able to connect to. After this is done the other instance that is not running the server needs to be able to search for nearby Bluetooth devices. After finding the desired Bluetooth device that we want to communicate with, it should be able to initiate a connection with that device. This will cause a communication link to be formed between the two different instances of the application. After the link is established, then both of the instances of this software will be able to send text messages to each other. There will also be an additional require that every time this software creates a connection with another instance of its self, the software must cause Bluetooth to perform the pairing process. If two devices have connected in the past, Bluetooth implementations will remember the link key that was used and store it. This will cause the two connecting device to skip the pairing process, but for this project we want the pairing process to be performed whenever possible. After a communication link is established the software then needs to be able to enter into a secure mode. This mode will represent the solution to an attacker being able to crack the pairing key used. Upon enter this mode, the two communication instances of the software will be required to generate and share RSA encryption keys. Then all further correspondences between the two instances will be encrypted using the RSA encryption. This will continue until secure mode is turned off, which at that point the software will revert back to sending plain text. When the user is finished with the connection they then can disconnect the two devices. If the software was hosting a server service then the user needs to be able to shut it down. The software should also keep a visual log of all the messages sent between its

instances displayed in plain text, and so the user has a record of what messages where sent.

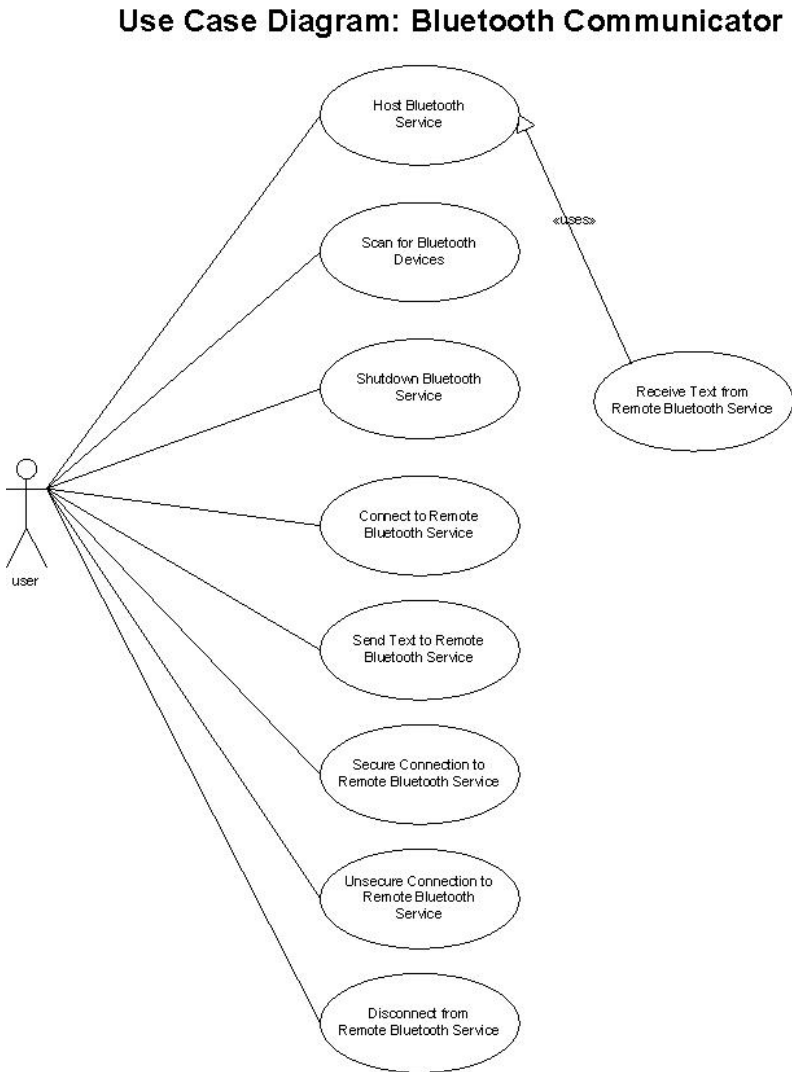
Software Package: Bluetooth Interceptor

This software package will represent a malicious user in a Bluetooth attack. The purpose of this software is to passively listen to two communicating Bluetooth devices and interpret the data they send between each other. To simplify this project this software will be designed to listen to the communications that are sent between instances of the software in package Bluetooth Communicator, above. This way you will know ahead of time what the format the data is being transmitted in. Also the above software provides a mechanism that forces pairing every time two devices are connected, and the capture of the pairing process is required for a passive Bluetooth listing attack to take place. This software requires a means to be able to monitor the communications between two different Bluetooth devices. The first method is to have special Bluetooth hardware that is capable of capable of capturing the Bluetooth radio signals and interpreting which two devices are communicating. This hardware costs thousands of dollars to purchase. However, it is also possible to take existing Bluetooth hardware and update the firmware on it to produce the same result as the specialized hardware that does this. This would be the preferred means to capture the traffic; however, it may be very difficult and time consuming to do this. As an alternate there is also tool available for the Bluez Bluetooth stack that allows you to capture the Bluetooth traffic sent through the Bluetooth stack on that machine. If this method is used that means one of the two instances of the communicating applications from software package A must be running on a machine that uses the Bluez Bluetooth stack. Once this software has a means to monitor the communications between two Bluetooth devices it will then monitor the traffic between two devices listening for key pieces of information. If it detects that a device pairing is happening it will then listen for the components of

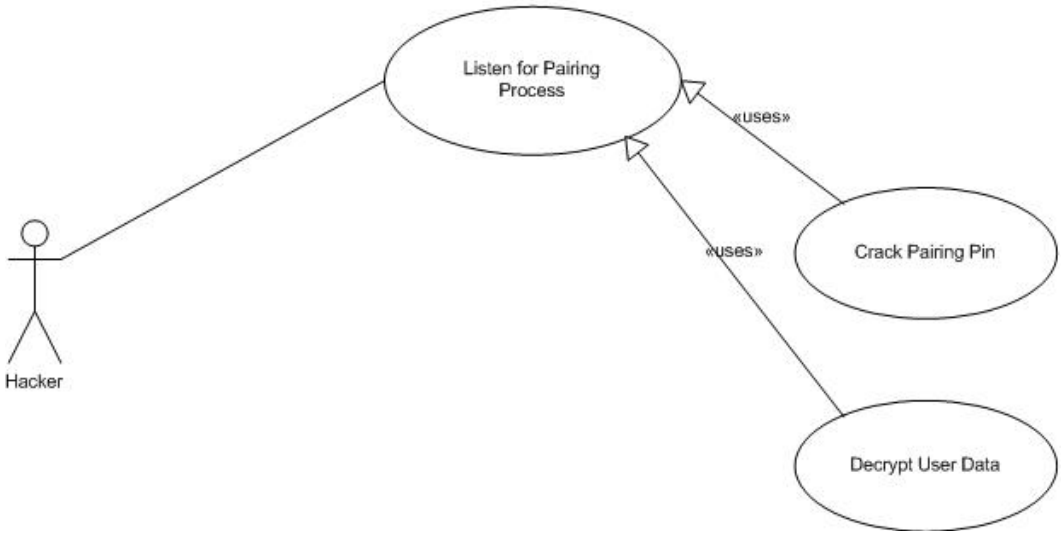
the pairing and store them. The software can then attempt to derive the pairing key from that information. If it finds the key the software should then decipher the communications that are being sent between the two legitimate devices and display it. If the two devices are communicating in the unsecure mode then it should be able to display the messages sent in plain text; however, if they are communicating in secure mode then the software would only be able to display the cipher that is being sent. It is possible that an offline attack on these messages could be performed but that is out of the scope of this project.

An iterative prototyping model was used to design this software that uses the: define, design, implement, and test paradigm. The use case documents in appendix B.2 were created in the define stage, and then later refined in subsequent iterations of the define stage. The class diagrams in appendix B.3 were created and refined in the design stages. The number of classes created and the line of code needed to implement this project were relatively few; however, this is intended. The proposed solution was designed to sit in between the RFCOMM and application layer with minimal impact on other systems.

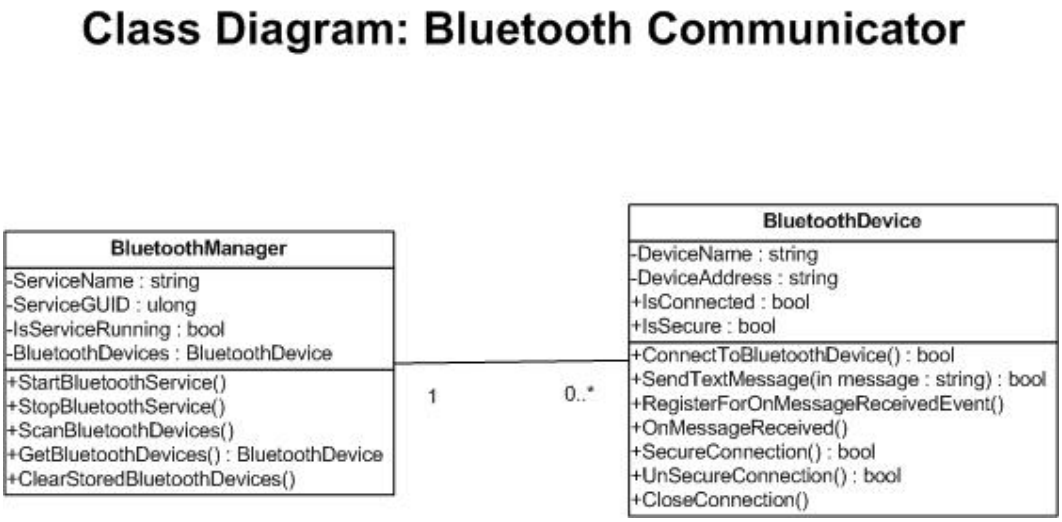
B.2. Use Case Diagrams

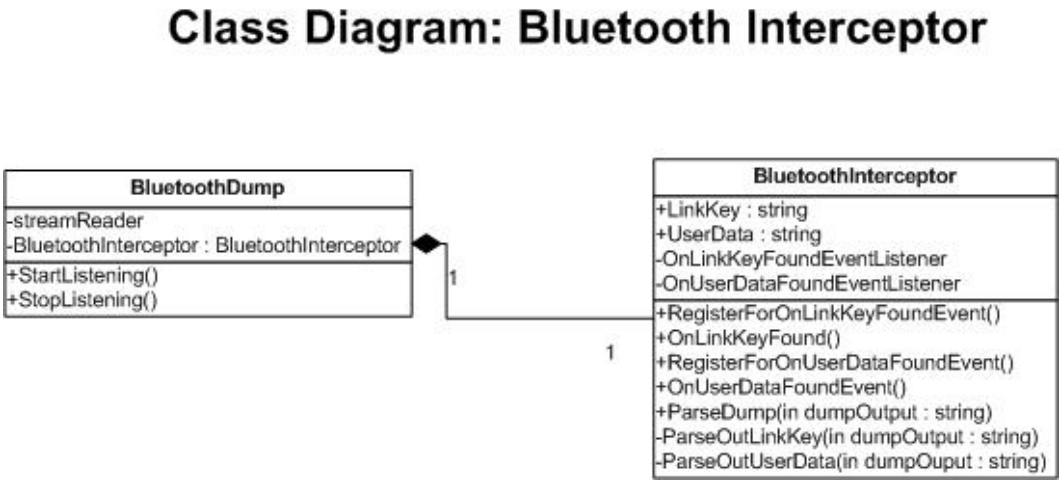


Use Case Diagram: Bluetooth Interceptor A



B.3 UML Class Diagrams





Appendix C: Formal Proof of Distributed Authentication Protocol

C.1. ProVerif Distributed Authentication Formal Proof: Part 1

```
free c.
(*
1. A -> S : { N_A, A }_pkS
2. S -> A : { N_A, m }_pkA, { N_A }_skS
*)

(* Public key cryptography *)
fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Host names. The server has a table (host name, public key), which we represent
by the function getkey. *)
fun host/1.
private reduc getkey(host(x)) = x.

(* Signatures *)
fun sign/2.
reduc checksign(sign(x,y),pk(y)) = x.
```

```
reduc getmess(sign(x,y)) = x.

(* Secrecy assumptions *)
not skA.
not skS.

private free mSADB.
query attacker:mSADB.
query evinj:endSparam(x) ==> evinj:beginSparam(x).

let processA =
        (* Choose the other host *)
        in(c,hostX);
        event beginSparam(hostX);
        (* Message 1: A -> S : { N_A, A }_pkS*)
        new Na;
        out(c, encrypt( (Na, hostA), pkS));
        (* Message 2: S -> A : { N_A, m }_pkA, { N_A }_skS*)
        in(c, (ms, hs));
        let (=Na,mASDB) = decrypt(ms,skA) in
        let =Na = checksign(hs, pkS) in
        if hostX = hostA then
        event endSparam(hostA).

let processS =
        (* Message 1: A -> S : { N_A, A }_pkS*)
        in(c,m);
        let (NY,hostY) = decrypt(m, skS) in
        (* Message 2: S -> A : { N_A, m }_pkA, { N_A }_skS *)
```

```
        out(c,(encrypt((NY,mSADB),pkA), sign(NY, skS))).

process new skA; let pkA = pk(skA) in
        out(c, pkA);
        new skS; let pkS = pk(skS) in
        out(c, pkS);
        let hostA = host(pkA) in
        out(c, hostA);
        ((!processA) | (!processS))
```

C.2. ProVerif Distributed Authentication Formal Proof: Part 2

```
free c.
(*
      3. B -> S : { N_B, B, U }_pkS
      4. S -> B : { N_B, _sSU}_pkB, { N_B }_skS
      5. B -> A : { N_C, B, U, _sSU }pkA
      6. A -> B : { N_C, _sAB }pkB, { N_C }_skA
*)

(* Public key cryptography *)
fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Host names. The server has a table (host name, public key), which we
      represent by the function getkey. *)
```

```
fun host/1.
private reduc getkey(host(x)) = x.

(* Signatures *)
fun sign/2.
reduc checksign(sign(x,y),pk(y)) = x.
reduc getmess(sign(x,y)) = x.

(* Shared-key cryptography *)
fun sencrypt/2.
reduc sdecrypt(sencrypt(x,y),y) = x.

(* Secrecy assumptions *)
not skA.
not skS.
not skB.
not sSU.

private free userU.
private free sAB.
query attacker:sAB.

let processA =
        (* Message 5: B -> A : { N_C, B, U, _sSU }_pkA *)
        in(c, m2);
        let (Nz,  hostZ, userZ, sSZ) = decrypt(m2, skA) in
        (* Message 6: A -> B : { N_C, _sAB, A }_pkB, { N_C }_skA *)
        out(c, (encrypt((Nz,sAB, hostA),pkB), sign(Nz, skA))).
```

```
let processB =
        (* Choose the other host *)
        in(c,hostX);
        (* Message 3. B -> S : { N_B, B, U }_pkS *)
        new Nb;
        out(c, encrypt((Nb, hostB, userU), pkS));
        (* Message 4: S -> B : { N_B, _sSU}_pkB, { N_B }_skS*)
        in(c, (ms, hs));
        let ((=Nb,=sSU), =Nb) = (decrypt(ms,skB), checksign(hs, pkS)) in
        (* Message 5: B -> A : { N_C, B, U, _sSU }_pkA *)
        new Nc;
        out(c, encrypt( (Nc, hostB, userU, sSU ),pkA));
        (* Message 6: A -> B : { N_C, _sAB, A }_pkB, { N_C }_skA *)
        in(c, (ms2, hs2)).

let processS =
        (* Message 3: B -> S : { N_B, B, U }_pkS*)
        in(c,m);
        let (NY,hostY, userY) = decrypt(m, skS) in
        (* Message 4: S -> B : { N_B, _sSU}_pkB, { N_B }_skS*)
        out(c,(encrypt((NY,sSU),pkB), sign(NY, skS))).

process new skA; let pkA = pk(skA) in
        out(c, pkA);
        new skB; let pkB = pk(skB) in
        out(c, pkB);
        new skS; let pkS = pk(skS) in
        out(c, pkS);
```

```
let hostA = host(pkA) in
out(c, hostA);
let hostB = host(pkB) in
out(c, hostB);
new sSU;
((!processA) | (!processB) | (!processS))
```