Bluetooth Attacks

INFORMATION IN THIS CHAPTER

- Bluetooth Technology
- Hacking Bluetooth
- Connecting
- Wholesale Sniffing
- Bluetooth Viruses

Bluetooth is one of those technologies that have become so common that it has become a part of our daily lives. It has become a solution to problems like driving and talking on a cell phone and introduced new and interesting marketing opportunities for attacks.

An interesting scam emerged that combined a Bluetooth attack with some creative usage of premium rate services. Its effectiveness has diminished recently, but the effects are interesting.

Imagine a commuter is traveling home by train every night. As he or she walks through the train station, he or she is concerned about getting home in time for dinner, not about getting scammed via the phone in his or her pocket. An attacker is sitting near the waiting area, scanning the air for advertising Bluetooth devices. He or she spots the phone in our commuter’s pocket and due to poor security choices is able to connect to it with a common or default PIN code. From there, the attacker instructs the phone to silently call out to a number, which stays connected until the commuter notices the phone call and dismisses it as accidental or the call is interrupted from a dead battery or service interruption. The attacker makes his or her money 1 month later when the commuter’s cell phone bill arrives. The number called was a premium rate line owned by the attacker that charges a high cost per minute ($3.99 or some such number). The victim’s phone called the number and incurred charges by the

\(^{A}\text{http://youtube.com/watch?v=cZRdnQ4g4NQ.}\)
minute on his or her way home for the night. Over a 45-min train ride, that charge can grow to hundreds of dollars that goes into the attacker’s pocket.

**BLUETOOTH TECHNOLOGY**

The need for Bluetooth was originally as a way to clean up our desks and decable our lives by being an alternative to the common RS232 cables that were needed to connect many devices to our computers. The ability to connect one device to several others simultaneously along with the ability to autoreconnect was a very exciting prospect for consumers tired of having to deal with many different cables for every device they owned. A single wireless method was needed to clean up the clutter.

Bluetooth is not actually one protocol, but it is actually a suite of protocols and functions wrapped up in a stack. Bluetooth operates in the 2.4 GHz ISM band along the same as 802.11b/g/n Wi-Fi; however, due to low transmit power and range, there is relatively little meaningful interference.

Bluetooth devices come in three classes. Each one is meant for specific applications and situations, but they are all the same in terms of compatibility. Most after-market Bluetooth adapters you can buy are Class 1 or 2. Class 3 is reserved for small devices like keyboards and a mouse that are sure to be near the computer they are connected to. Table 3.1 shows the relative power and range of each class.

Bluetooth has also undergone some revisions and multiple versions of the protocol exist. Bluetooth 1.2 and 2.0 + Enhanced Data Rate (EDR) are the most common in the marketplace, and you are unlikely to run into any of the 1.0, 1.0 B, or 1.1 devices except perhaps on some legacy and early devices.

Bluetooth 1.2 supports approximately 725 Kbps throughput, which is enough for things like mouse, keyboards, printers, headsets, and so on. Revisions to the standards culminating in Bluetooth 2.0 included EDR, which bumps the maximum throughput to 3 Mbps on compatible devices for larger data such as photos or music files.

Recently, the Bluetooth Special Interest Group (SIG) adopted the core specifications for Bluetooth 3.0. This specification works with the 802.11 radios in many products for a maximum throughput of 24 Mbps. Actual products based on the specification should arrive on the market in the beginning of 2010.

<table>
<thead>
<tr>
<th>Class</th>
<th>Transmit power</th>
<th>Effective range</th>
<th>Typically found</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 1</td>
<td>100 mW</td>
<td>100 m</td>
<td>External USB adapters with external antenna</td>
</tr>
<tr>
<td>Class 2</td>
<td>2.5 mW</td>
<td>10 m</td>
<td>Internal laptop adapters</td>
</tr>
<tr>
<td>Class 3</td>
<td>1 mW</td>
<td>1 m</td>
<td>Cell phones, headsets, keyboards</td>
</tr>
</tbody>
</table>
Bluetooth devices are connected through a process called pairing. The pairing process usually involves one device searching for other devices in the area and then selecting the device to partner with based on its BD_ADDR (similar to a computer Media Access Control [MAC] address) or a logical name. Once pairing is completed, the devices are now considered “bonded.”

Depending on the action being taken, a PIN for the device being connected will be required to complete the process. The PIN acts as a password in an encryption scheme between the two devices and is used to generate a link key. This key is used to secure communications between the devices and also to authenticate devices in the future. Should a device want to reconnect in the future, the devices check if each other’s BD_ADDR is already bonded, and if it is, they assume that both have the link key still stored and will immediately begin communicating. This skips the PIN entry process and allows for automatic reconnection when in range. If one or both of the devices have forgotten the link key, the pairing process is restarted, and the PIN is required to be entered again.

Even before they are connected, devices communicate their capabilities to each other. Just as not every host on the Internet is a Web server or a Secure Shell (SSH) server, not every Bluetooth device talks on every service available. Through a protocol called Service Discovery Protocol (SDP), the devices announce to each other capabilities.

EPIC FAIL

While Bluetooth-enabled phones are now commonplace throughout Europe, the adoption rate has been slower in North America. Early on, many North Americans were unaware of all the capabilities a Bluetooth-enabled phone offered since they have never actually had one with all the features. While not indicative of all models sold, many North American cellular providers selling Bluetooth phones were selling ones that had features disabled. Things like OBEX data transfer are disabled in the firmware and unavailable. The official reasoning was as a security precaution due to the issues described in this chapter, and some argue though that it is meant to preserve a revenue stream.

Most phones now have a camera built-in and the capability to use custom ring tones. These features are a great source of income for cellular providers. If you take a picture with your phone, you have to transfer it off if you want to upload it to the Web or share it with others. This requires an expensive data cable (sold by the provider), expensive removable media (if available on the phone, and sold by the provider), or uploading over the cell network via MMS or data, which costs airtime or additional charges.

The ability to pair a Bluetooth phone to a computer and download pictures or upload ring tones without additional charge or even wirelessly tether is quite appealing to the consumer. When so many computers come with Bluetooth capability, the user has everything he or she needs to transfer data without additional charge. This is why many cell phones are not capable of more than operating a Bluetooth headset or as a virtual dial-up modem.

In 2004, Verizon released the Motorola V710 phone. It was advertised as Bluetooth capable but controversy arose over the fact that Verizon failed to mention that most of the Bluetooth profiles were disabled via firmware, and the phone was not quite as advertised. A class action lawsuit was launched and eventually settled in September 2005.

The moral is that if you are buying a Bluetooth-enabled phone, make sure to check to see if all profiles are available if you want them.
what services they support. A computer may support all possible profiles such as networking, file transfer, and others, whereas smaller devices only support a few or a single profile. A headset, for example, does not support networking and only has the audio profile available. This announcement of profiles allows devices to filter by capabilities, such as a phone looking only for headsets or a computer looking for a printer rather than being flooded with unnecessary devices.

Most often, the process is fairly painless to the end user. Some profiles can operate without encryption, say if you want to send over contact information from one phone to another without complicating matters with PIN numbers. Those that require encryption only require simple PIN numbers to complete the process. The addition of remembering bonded devices and automatic reconnect makes it even easier.

**HACKING BLUETOOTH**

With the widespread adoption and convenience of Bluetooth device comes the inevitable implementation problems that cause unexpected things to happen.

Most Bluetooth-based attacks are based on a simple and common flaw. Users often are very poor at reading documentation, at understanding risks and threats, and generally at changing defaults. Most attacks revolve around users not changing the default settings on their devices. That, coupled with poor user interface (UI) design, creates situations where users are unaware something bad is happening or about to happen. As well, documentation until recently did not fully explain the risks of certain actions.

**Bluetooth Discovery**

As with most attacks, the first thing to do is to find a target. Most often, Bluetooth attacks are against targets of opportunity (that is, not targeted). In the case of Bluetooth, its design assumes that devices interact with one another occasionally (that is the whole point of the technology). The need for these devices to find one another easily is a requirement of this. This allows legitimate users to find the device they are seeking, but also allows a nearby attacker to find those same devices and silently interrogate them to find out if they are suitable to attack.

The Bluetooth discovery process involves both parties in the pairing process. A device seeking to connect sends out a broadcast to the immediate area. Devices in the area, if they are set to be discoverable, then respond with the BD_ADDR of the device. From there, the querying device can use the SDP to discover all the profiles (that is, services) the device is offering (and vice versa). Simply put, a cell phone sends out a query, the headset responds saying it supports the hands-free profile, and the cell phone now knows to treat this remote device as a headset and not a dial-up adapter or some other profile.

Most cell phones, PDAs, or other Bluetooth devices have the capability to scan for discoverable devices. Even with this very simple interface, you can glean some
interesting information. Many phones by default have the make and model of the phone listed as the name of the device when queried in a scan. This automatically makes life easier for attackers because they can immediately see what device it is and determine if there are any specific procedures needed to do bad things to it. Alternatively, when people do change the default settings, they very often change them to their name or some other very easy identifier like “Bob’s phone” or “Alice’s Blackberry.” If an attacker is specifically looking to target Bob, then Bob has made his phone obvious among the others detectable in range and thus an easier-to-find target. By the same token, if the attacker knows Bob has a Nokia phone and sees only one device named Nokia 6330, then it’s pretty obvious which one is Bob’s.

The BD_ADDR, much like MAC addresses for network cards, has a known format to follow. They are 48-bit identifiers in a HEX format. The first 3 bytes are assigned to specific manufacturers. Like MAC addresses for network cards, the Institute of Electrical and Electronics Engineers (IEEE) assigns blocks of addresses to manufacturers to embed in their devices. The last 3 bytes are the unique address of the device. Since the assigned addresses are public, if the device name is not helpful, the BD_ADDR is not hidden and can be used to determine manufacturer. Anyone can query the OUI database and see what manufacturer made the device and adjust your attack accordingly. The IEEE has a query page up at http://standards.ieee.org/regauth/oui/index.shtml, and you can also download a copy of the whole database for integration into your own application or for local lookup.

### NOTE
While manufacturers are assigned addresses in blocks that can contain millions of device addresses, that doesn’t mean that all of them are used in the real world. Manufacturers may only use a few hundred thousand and then move on. The BNAP project aims to collect device addresses and see what actual manufacturer prefixes are in use. This greatly reduces the number of possibilities if we are searching address space since we can eliminate addresses not in use. The BNAP project’s list of address prefixes and the manufacturers are available at http://802.15ninja.net/bnapbnap/.

Applications like BTscanner for Linux or Windows (www.pentest.co.uk/cgi-bin/viewcat.cgi?cat=downloads) or Bluescanner for Windows (https://labs.arubanetworks.com/) can automate the process of scanning for devices, interrogating and cataloging the results. This allows attackers to sit and observe the results much like Kismet (www.kismetwireless.net) and Netstumbler (www.netstumbler.com/downloads/) did with Wi-Fi networks.

There have been several projects over the years to quantify the number of Bluetooth devices in any given area. In early 2006, F-secure along with secure networks created the Bluebag, a mobile Bluetooth detecting rig including nine Bluetooth adapters built into a hard-sided wheeled suitcase. They wheeled the case through airports, shopping malls, and even a security conference. In 23h of scanning at various locations, they discovered 1405 unique discoverable devices, many of
which were advertising various profiles such as OBEX file transfer and headset
capabilities. These devices may be susceptible to various attacks. The final report is
interesting reading and is available at www.securenetwork.it/bluebag_brochure.pdf.

Another project to catalog Bluetooth devices has been going on since 2007 in
the Netherlands. The Bluetooth tracking project (www.bluetoothtracking.org) has
set up multiple sensors in various parts of the Netherlands and even Paris, France,
that continually monitor for discoverable devices and record the information in a
database. Since then, they have collected over 600,000 devices and generated some
interesting usage statistics. They also have been able to calculate the speed at which
a device is moving. One sensor detects a device, another a few miles away detects it
again – from that you can extrapolate the speed at which someone was moving and
what direction.

Some devices, as a security measure, do not respond to probes or only have lim-
ited window of time where the device is discoverable. These devices can cause a bit
of a problem for the attacker; however, they are not impossible to detect with a little
coercion and brute force.

Devices made nondiscoverable still can connect to other devices, except that the
other device needs to know the BD_ADDR. If Alice wants to connect to Bob and
Bob’s phone is not discoverable, Alice needs to know Bob’s BD_ADDR before she
can connect. If she does not know it, Bob needs to make his device discoverable or
tell Alice his device’s BD_ADDR, and she can enter it (if possible) to specify what
to connect to. When Bob’s phone sees a directed request to connect to its address, it
allows the connection to continue.

From an attacker’s perspective, this makes it much harder to attack Bob’s phone
since it won’t respond unless the attacker knows the BD_ADDR. It’s unlikely Bob
would share that with just anyone, but by using some basic information, we can
potentially find out Bob’s BD_ADDR.

Programs like Redfang (www.net-security.org/software.php?id=519) written by
Ollie Whitehouse allow you to force connection requests through a particular address
space. If we know Bob’s phone make and model, we can either scan a similar model
to determine its address range or search the IEEE database or BNAP database for that
manufacturer known addresses. This will get us the likely first 3 bytes of the address.
The remaining bytes will have to be brute forced sequentially until a response is
received from Bob’s phone. Since each query can take around 10s to be thorough in
waiting for a response, the process can take a while, from minutes to days depending
on the type of device and the address space being searched.

**CONNECTING**

Once a device has been found, the device can begin connecting. In most cases, this
involves an encrypted link be established before higher level communications begin.
While there are variations, most often the device requesting the pairing is required
to enter the secret PIN code from the other device. The theory is that only those who
know the PIN of the remote device can connect. This may mean asking the owner of the other cell phone or just the fact that you own the device means you set its PIN and know it. Other variations include having the second device enter a code which has to be entered on the first device as well, thus requiring user interaction and avoiding default PINs. Once the PIN is entered, the devices handshake, and a link key is established, and now the devices are bonded and trust one another. This is an over simplification of the internals of the process, but the main issue here is not one of internals but of externals.

The most common attacks on Bluetooth devices are due to default PIN numbers. Many devices come preset from the factory with a PIN that the user is asked to change. Usually those instructions are buried in the manual and are ignored by the user. Other devices, such as headsets, have no interface in which to change the PIN, so their PINs stay static. The most common PINs are 0000, 1234, and 9999. A quick Google search for a specific device will turn up the manual, which will usually contain the default PIN, right next to instructions on how to change it. This situation leaves plenty of room for an attacker to play. The most famous of these attacks involving default PINs concerns Bluetooth headsets.

**Carwhisperer**

It’s hard to escape them. Bluetooth headsets have become incredibly commonplace in daily society. The adoption rate has recently skyrocketed due to local laws that ban using a hand-held cell phone while driving. For others, they have been elevated to fashion accessories. Needless to say, as long as Bluetooth phones remain popular, these devices will continue to flourish.

Headsets are also related to the increasingly popular in-car Bluetooth hands-free devices. Occasionally integrated in vehicles by the manufacturer, and also available as after-market units, these systems take advantage of the car’s built-in stereo system to allow the driver (or passenger, as it may be) to hear the audio from the phone while keeping both hands free (hopefully to drive). A small microphone is used to relay voice of the driver to the phone and to the other party – essentially, the same function as a headset, just not as fashionable, and not the easiest to wear on your head. For the remainder of the chapter, both in-car systems (manufacturer installed and after market) and wearable headsets will be discussed as the same thing since they are functionally identical.

Early Bluetooth security research noted that the usage of default PINs was going to be a problem since users, given the option, often not change a default setting unless forced. This led to carwhisperer being developed to exploit this.

In July 2005, carwhisperer was released by Martin Herfurt of the Trifinite group (http://trifinite.org/trifinite_stuff_carwhisperer.html). This software took advantage of the default PIN situation to provide a fun and sometimes frightening prank.

Carwhisperer is a series of scripts along with the actual application. The first cw_scanner probes for devices with the device classes common to headsets and hands-free devices. Once one has been found, the carwhisperer program initiates a connection
to the device, and with the help of the cw_pin script, selects the appropriate default key based on the manufacturer prefix for the device’s BD_ADDR. Once the connection is made, the attacker can record audio from the device’s microphone or inject audio of his or her choice into the speaker. One can imagine the fun that can be had with the latter.

Once connected, the attacker can continue to listen and inject as long as they remain in range and the device connection is not broken. Carwhisperer will not work on devices already paired and communicating, as headsets only support one connection at a time. While this may seem a problem, really it is not. Many vehicles support the feature, but the person driving may not have his or her phone paired at the time. The car’s system defaults back into discoverable mode, waiting for a connection. With headsets, there are specific places where headsets will be on but not paired. Favorite locations are movie theaters and airports since both locations require you to turn off phones, but people forget to turn off headsets.

**WARNING**

Be aware that airports, airlines, and flight crews are very strict about the rules regarding mobile devices being used at the wrong time. Some airlines and airports allow devices to be used during taxiing after landing, others do not. Still depending on the airport and the port of call, mobile devices may be forbidden from use until well after leaving the plane. Violating any of those rules can bring about fines and legal entanglements with the authorities.

The best location to do any guerilla Bluetooth auditing is as you wait to get on the flight and the passengers from the arriving plane disembark and turn on their devices – best to not tempt fate, safety, and fines by scanning while in flight.

While carwhisperer’s usage in pranks seems obvious, a more devious use is in low-cost industrial espionage. Often in businesses, the policy for high-level meetings is to turn off phones, either for security reasons or to avoid distractions. Often people will turn off their phone but not their headset. Turning off the phone will break the pairing, and the headsets will return to a state seeking something to pair with. This allows an attacker to turn the headset into a listening device (bug) in the meeting and potentially divulging sensitive information.

The most obvious and easy solution is to not use Bluetooth headsets at all. However, this is not always an option and neither is changing the PIN on the headset since there is no interface to do so. The most effective way to avoid carwhisperer eavesdropping is to simply turn off the headset when not in use and to not wear it when not in a conversation (it’s hard to record audio when it’s in your pocket). Hands-free systems, in cars, can disable the feature when not in use or not paired.

**Bluebug**

The opening of this chapter described an attack involving Bluetooth being used to force other phones to make calls to expensive numbers owned by the attackers. Typically these are not one specific attack, but more a description of the end result, however
achieved. Bluebug is the name given to the attack goals; however, the methods can vary. Originally named by the Trifinite group, it describes any attack that gains access to the Bluetooth device through an RFCOMM (another Bluetooth service) channel connection. Tools like Bluebugger (www.remote-exploit.org/codes_bluebugger.html) and Bluesnarfer (www.alighieri.org/project.html) automate the process that can be performed manually with some basic tools included with most Linux distributions.

These attacks are possible because of poor implementation by manufacturers or through undocumented debug or diagnostic services being left open. Some Bluetooth devices, most often mobile phones, expose RFCOMM ports (wireless serial ports) to the world and do not require authentication. An attacker can locate and scan a device for open RFCOMM channels and connect via a serial terminal. Depending on the phone’s inherent vulnerabilities, the attacker can have nearly unrestricted access to the phone’s contents and services, including reading SMS messages, reading phone book entries, issuing Hays modem (AT) commands, and allowing the attacker to perform such actions as setting up call forwarding or initiating calls. Due to the fact that the attacker is interacting directly with the phone, the UI often will not react and give any indication that the phone is doing anything.

To use the example at the beginning of the chapter of a commuter being attacked as he or she waits for the train, the attack can be accomplished with little more than built-in utilities and tools included with most operating systems. Obviously scripts and purpose-built tools can make this easier, but it does nothing to change the effect of the attack.

The attacker, using btscanner or some other application, would scan for discoverable Bluetooth devices. The BD_ADDR would show that the victim’s phone is a Nokia. The name of the device is Nokia 6310i, which tells the attacker this is likely a vulnerable phone. Using sdptool (a utility included with the Linux Bluez protocol stack), the attacker can browse the advertised list of services and see if anything looks interesting. He or she discovers that RFCOMM channel 17 is open, and uses Bluebugger or Bluesnarf, or even the included rfcomm tool and serial terminal, to connect to the port and begin issuing commands. From there, the attacker can issue a command like “ATDT19001234567”, which would tell the phone to dial a premium rate number that charges a large, per minute charge. If all goes well, the phone will connect and continue the call long after the victim has left the range of the attacker.

The worst case (or best case, depending on which side you are on) that will happen is that the target device will not be vulnerable and will require pairing before allowing connections to the RFCOMM port and may trigger a pairing request on the victim’s phone, which will often be ignored or dismissed. Depending on the device though, if the victim is using a default PIN and his or her device does not require user interaction to pair, then all bets are off as the attacker can now connect anyway.

This attack causes the victim’s service provider to act as the collection agent. The victim will have a hard time disputing the charge to the service provider since the call did originate from his or her phone and the idea of an outside attacker is often a hard sell. If the attacker is very creative, he or she will set up the phone line to have a very embarrassing name (some phone sex line or something of the like), so the victim is embarrassed enough at having to defend himself or herself that he or she didn’t call
“Madame Whipsalot fun time party line” that he or she will just pay the charge to
make the issue go away.

**TIP**

Some phones have open RFCOMM ports but are not advertised via SDP. These ports are
leftovers from diagnostics used by the manufacturer and can be accessed by connecting
to the channel directly. Depending on the phone, there are 60 channels available for
simultaneous use. An attacker can just step through all ports to see if any respond. This is
a time-consuming process, and it is more likely that he or she already knows which phones
have these back channels open. If you are working to assess a Bluetooth device, it is worth
stepping through all RFCOMM channels, just to be sure.

It is worth noting that a direct RFCOMM connection is not required to extract
or abuse a target device. Other Bluetooth profiles can be abused to read information
from devices. Profiles like OBEX push, typically used to push contacts from one
device to another, can be exploited since the connection is actually two-way. Instead
of pushing a contact, an OBEX GET request is made for a known filename. Files
such as telecom/pb.vcf, which is the device’s phone book, would be most useful to
extract. This type of connection often does not require authentication. Some devices
also can be caused to crash by pushing a vcard (a contact) with a very long file name,
causing a buffer overflow and, possibly, crashing the phone.

At this point, it must be pointed out that many of these attacks have been miti-
gated through firmware upgrades and improved UI design. From about 2003 to 2007,
these attacks were widespread as the technologies were new, and the user base was
unaware of the risks. Many phones now are not discoverable, and most manufactur-
ers have improved their implementations so as to not expose users to such risks.

That being said, there are always new devices coming on the market with
Bluetooth capability. There may be an occasion where a manufacturer slips up and
exposes a vulnerability. From the perspective of an IT or security manager, it would
be a good idea to evaluate all Bluetooth devices and their peripherals for obvious
vulnerabilities before allowing them in production environments.

From an auditing standpoint, tools like Bluetooth Stack Smasher (www.secuobs.
com/news/05022006-bluetooth10.shtml) are invaluable as they have fuzzing capabil-
ities to allow for discovery of previously unknown issues of security and stability.

**WHOLESALE SNIFFING**

In 2007, n.runs released Btcrack, a Windows program to crack Bluetooth link keys
(www.nrruns.com/_en/security_tools_btcrack.php). While this would seem to be a
death sentence for Bluetooth, there is a problem. Sniffing the packets necessary to
enable the crack to occur is a lot harder to catch than you would think.
As with any wireless technology, it all boils down to the fact that it is a radio, and radios operate in a shared medium, the electromagnetic spectrum. As such, its presence is impossible to hide. In order to communicate, it has to betray its presence. In the case of Bluetooth, this is the 2.4 GHz spectrum, the same as 802.11b/g/n networks; however, its operation is markedly different from Wi-Fi that creates some unique issues.

Unlike Wi-Fi, there is no monitor mode equivalent in Bluetooth since it’s a suite of protocols rather than one cohesive package. When devices connect to one another, the master device of the piconet sets the hop scheme to all slave devices from its internal clock. Bluetooth operates using frequency hopping spread spectrum (FHSS) across 79 channels, each 1 MHz wide and hops at 1600 hops/s, which means that if we don’t know the clock of the master, there is no easy way to follow the hopping sequence of the piconet. We won’t be able to get all the packets. As well, it is not possible on a standard Bluetooth device to listen to all channels simultaneously, nor is it practical to have 79 Bluetooth radios each listening to a single channel.

**NOTE**
There are Bluetooth protocol analyzers on the market that do allow you to sniff all traffic. Typically these devices require the BD_ADDR of the master and slave devices and must follow the communication from the beginning, and even then, they can be tricky to get working right. These devices are not able to just walk up and listen in on random Bluetooth conversations like you can with Wi-Fi and monitor mode. These devices also cost thousands of dollars and are out of reach for most attackers and researchers alike.

An interesting thing about some of these analyzers, though, is that they often use the same hardware as consumer radios but different firmware. It was discovered in 2007\(^8\) that some companies used the same radio chip in the hardware portion of their analyzer as consumer radios, the only difference being a single digit on the USB identifier and the firmware on board. It did not take long before people figured out that they could pirate the software portion of the analyzer, modify the firmware on cheap consumer Bluetooth dongles, and have access to an analyzer. Unfortunately for research, this was not of much help since, as noted above, you needed to know a lot more about the connection than a random attacker would in order to sniff all the traffic. However, this did point out that the hardware was capable, that the rest of the issues were software, and that it may be possible to build an open source protocol analyzer.

In order to sniff Bluetooth, we need to listen to the raw baseband signaling and all channels simultaneously to capture whatever traffic may be going by. This requires more than what a standard Bluetooth adapter can do. However, that does not mean that there is no hope.

The GNU Radio project was started to create a software-defined radio to allow people to freely hack the radio spectrum. Instead of specialized hardware to transmit and receive on a specific frequency, the GNU Radio went about designing a system with an field programmable gate array that could be reconfigured to any purpose.

\(^8\)http://remote-exploit.org/research/busting_bluetooth_myth.pdf
The end result was the Universal Software Radio Peripheral (USRP), the companion hardware to the GNU Radio software, which allows researchers to reconfigure the device to behave like any radio device they need.

The USRP has some limitations for this end. The USRP2 (second-generation hardware) has a maximum receive bandwidth of 25 MHz, meaning that it can only read 25 Bluetooth channels at once, so you would need four USRP2s in order to scan all channels and capture everything.

Various projects like the GR-Bluetooth project (http://gr-bluetooth.sourceforge.net/) aim to turn the USRP into a baseband receiver capable of scanning all Bluetooth channels and capturing all traffic through various techniques. The project is ongoing and continually posts new code to its Web site.

In order to capture the necessary packets to crack the link key (the PIN is unnecessary to crack since it’s used to derive the link key, and if you know the link key, reconnection is possible without victim interaction), there are a couple of options. The first involves impersonating the slave device in the piconet. The attacker determines the BD_ADDR of both devices. He or she then changes the BD_ADDR of his or her adapter to the BD_ADDR of one of the slave devices. The new slave tells the master that it has lost the link key, causing the master to request a new link key from the real slave. This is then captured and fed into Btcrack and out pops the link key with which can be used to decrypt the traffic captured from then on.

**BLUETOOTH VIRUSES**

Viruses are incredibly common today and are part of the background noise associated with the Internet and computing. While news of a new virus on a desktop computer rarely gets attention anymore, it is a surprise when it someone’s phone that gets infected, particularly through Bluetooth.

Bluetooth has already been an infection vector for certain Bluetooth-borne viruses. The most well known of these viruses was the Cabir worm. In 2004, this worm surprised the world since it attacked cell phones instead of desktops. Targeting phones using the popular Symbian OS (SymbOS), the virus spread over Bluetooth connections automatically. Once a device was infected, no action was needed by the user to spread it; however, to be infected required a fairly large amount of user interaction. In this case, users would receive a request to transfer a file over Bluetooth from an infected device. They would then have to accept the transfer despite some devices warning against such actions. Once accepted, they would have to run the file, which would install the virus and begin propagation. The virus would then seek out other discoverable Bluetooth devices and continue the process.

Despite this obvious complexity, users still installed the virus and contributed to the infection. Often the infected device would repeatedly send transfer requests to the targeted device requiring the user to keep hitting “no.” Often, the user would get frustrated with the incessant requests and would hit “yes” to find out what the fuss was
about. Fortunately, various errors in the worm’s programming limited its infection attempts to only one other device, keeping its propagation small and manageable.

This virus and others contributed to a small but growing industry for mobile device antivirus solutions. While not as common as its Internet-borne cousins, Bluetooth-based malware continues to be a potential issue that users have to understand and manage.

**SUMMARY**

Despite everything that you hear about Bluetooth security and its failings, it is not as bad as it seems. Many of the failings were initially due to poor implementations by manufacturers. As time has progressed, manufacturers have caught on about how to implement Bluetooth devices properly, avoiding these issues. However, with so many new devices coming on the market every day, it is very likely that issues will potentially creep back in, particularly from manufacturers new to making Bluetooth devices.

Care should be taken in any environment to ensure that users understand the risks associated with their Bluetooth devices. At the least, cursory testing should be done on any new device to make sure it does not have any glaring holes.

Ensuring that default PINs are not used and that Bluetooth functionality (at least discoverability) is properly configured can go a long way to limiting the impact of Bluetooth threats.